Université de Mons
Faculté des Sciences
Institut d'Informatique
Service d'Algorithmique

Conception d'un lecteur de musique intelligent basé sur l'apprentissage automatique.

Mémoire réalisé par **Xavier DUBUC**
en vue de l'obtention du diplôme de Master en Sciences Informatiques.
Directeur : Mr Hadrien MÉLOT

UMONS
Université de Mons

Faculté
des Sciences

Année académique 2011-2012

Remerciements

Je remercie mon directeur, *Hadrien Mélot*, pour sa disponibilité, son aide précieuse et la relecture du présent document.
Je remercie également mon frère, *Jérôme Dubuc*, pour la correction orthographique et syntaxique ainsi que mon ami, *Jérôme Descamps*, pour la relecture et l'aide apportée pour rendre ce mémoire plus accessible.
Finalement, un remerciement spécial à ma fiancée *Morgane Capelleman* pour ses talents d'infographiste bien utiles pour certaines images et pour le rendu graphique de l'application.

Table des matières

Introduction

La musique fait partie de notre quotidien, que l'on en écoute ou que l'on en joue. L'écoute se fait notamment via l'utilisation de logiciels de lecture de fichiers musicaux. Depuis quelques années, ces logiciels sont devenus plus que de simples lecteurs. En effet, ils se rapprochent de plus en plus de l'utilisateur en offrant des fonctionnalités lui permettant de n'intervenir que très peu et de lui rendre la lecture de musique des plus agréables.

Ces fonctionnalités sont rendues possibles via l'utilisation d'un concept de l'intelligence artificielle en pleine expansion : l'*apprentissage automatique*. Cette discipline consiste à développer une intelligence artificielle capable d'améliorer sa manière de fonctionner grâce à des exemples fournis ou à ses expériences.

Parmi ces fonctionnalités, on trouve, entre autres, la confection de listes de lecture selon l'humeur ou les goûts de l'utilisateur (Sony SensMe, Jukefox, ...) ou encore la proposition d'achats de nouvelles musiques qui sont susceptibles de plaire à l'utilisateur (comme Apple Genius). La majorité de ces fonctionnalités, pour ne pas dire toutes, sont orientées vers les musiques écoutées par l'utilisateur et non vers ses actions. Dans le cadre de ce mémoire, le logiciel d'apprentissage automatique qui est développé se base principalement sur les actions de l'utilisateur. De cette manière, si, par exemple, l'utilisateur passe une musique, le logiciel comprendra que cette musique n'était pas la bienvenue à cet instant.

Ce mémoire passe en revue les différentes techniques d'apprentissage automatique en s'attardant particulièrement sur les techniques utilisées dans celui-ci. Tout ceci est traité dans le chapitre 1. Dans le chapitre 2, il établit l'état de l'art du domaine de la lecture de musique aidée par l'apprentissage automatique. Il décrit finalement, dans le dernier chapitre, les étapes de création d'un lecteur de musique "intelligent" qui nommé "**SmartPlayer**" au travers de sa conception et de son implémentation dans le langage JAVA.

Chapitre 1

L'apprentissage automatique

Ce mémoire repose entièrement sur l'*apprentissage automatique* (ou *machine learning*). Cette pratique est l'une des nombreuses branches de l'*intelligence artificielle*, qui est elle-même une des branches en pleine évolution des sciences informatiques. Ce chapitre commence par introduire brièvement l'intelligence artificielle pour ensuite s'étendre sur l'apprentissage automatique ainsi que sur les différents types d'apprentissage utilisés et leurs applications.

1.1 L'intelligence artificielle

Cette section se base sur "Artificial Intelligence : a Modern Approach"[1], le livre de Russel et Norvig, la référence en la matière. Elle aborde très brièvement les bases de l'intelligence artificielle ainsi que quelques applications. Les lecteurs désireux d'en apprendre plus sur l'intelligence artificielle devraient donc se tourner vers ce livre [1].

Définition 1.1. *L'**intelligence artificielle** est la branche de l'informatique s'intéressant à développer des logiciels capables d'agir de manière autonome et rationnelle. En d'autres mots, ces logiciels sont censés effectuer les meilleures actions possibles parmi celles qui lui sont disponibles.*

L'intelligence artificielle est souvent perçue comme une discipline s'intéressant à créer des robots ou des logiciels capables de battre des humains ou de jouer au mieux à des jeux (ou plus rarement à des sports). Ceci ne représente cependant qu'une partie de ses applications réelles. En effet, on peut retrouver dans ces applications des logiciels simples comme ceux qui sont embarqués dans les aspirateurs automatiques ou des logiciels permettant de découvrir voire de démontrer des théorèmes mathématiques. On peut également retrouver des logiciels plus évolués comme ceux intégrés dans les voitures sans pilote de Google [2] ou encore des logiciels capables d'automatiser la planification logistique et l'organisation des transports d'une entreprise (comme DART [3] utilisé lors de la seconde guerre mondiale par les Américains).

L'agent. Un logiciel d'intelligence artificielle est appelé *agent*.
Celui-ci est considéré comme une entité agissant de manière autonome sur son *environnement*. Un agent de manière générale est composé de plusieurs parties distinctes :

- **le(s) senseur(s)** qui lui permet(tent) de **sentir l'environnement** et donc de se rendre compte de l'*état* dans lequel celui-ci se trouve ;
- **l'(s) actuateur(s)** qui lui permet(tent) d'**agir sur l'environnement** et donc de modifier son état ;
- **la mémoire** qui lui permet de se **souvenir** de ses perceptions (et parfois même des actions qu'il a effectuées) ;
- **l'intelligence** qui consiste en le processus visant à "transformer" les perceptions obtenues par le(s) senseur(s) en actions à effectuer par l'(les) actuateur(s).

De manière formelle, un agent est donc considéré comme un logiciel consistant à transformer ses entrées (**perceptions**) en sorties (**actions**) en suivant une certaine **intelligence** et en se basant sur ses **perceptions/actions antérieures** si nécessaire.

1.2 Qu'est-ce que l'apprentissage automatique ?

La branche de l'intelligence artificielle dont fait l'objet ce mémoire s'appelle l'"apprentissage automatique" et celle-ci est développée dans cette section. Cette section démarre par une définition formelle de cette discipline pour ensuite en expliquer les fondements et en exhiber quelques applications. Mis à part les parties concernant l'apprentissage par renforcement et l'apprentissage non-supervisé, toute cette section est basée sur la référence AIMA [1].

Définition 1.2. *Un agent **apprend** s'il augmente ses performances sur des tâches futures après avoir fait des observations sur son environnement.*

Définition 1.3. *L'**apprentissage automatique** est la discipline informatique visant à doter des agents de la capacité à **apprendre** de leurs expériences ou à partir d'exemples et ainsi adapter leur comportement, leurs réponses à l'évolution de leur environnement.*

Exemple 1.2.1. *Un logiciel servant à classer les mails reçus par un client de messagerie électronique en spam/courrier pertinent, pour être efficace, doit avoir recours à l'apprentissage automatique. En effet, lorsqu'un utilisateur spécifie à son client de messagerie qu'un mail reçu est un spam ou qu'un mail classé comme spam est en fait un mail pertinent, celui-ci apprend de cette action en adaptant sa politique de tri. Il l'adapte pour qu'à l'avenir les messages semblables à celui concerné par l'intervention de l'utilisateur soient correctement classés. Sans apprentissage, il ne cesserait de faire les mêmes erreurs.*

1.2.1 Utilité

Il pourrait être intéressant de se questionner sur l'utilité, le besoin qui peut exister d'un agent qui apprenne. Si l'agent peut évoluer, pourquoi ne pas le programmer directement dans sa version la plus évoluée et ainsi éviter l'utilisation de l'apprentissage ? Il y a en fait trois raisons à cela :

1. les concepteurs ne peuvent anticiper toutes les situations dans lesquelles se retrouvera l'agent ;
2. les concepteurs ne peuvent anticiper tous les changements dans le temps ;
3. les concepteurs n'ont parfois même pas l'idée de comment programmer l'agent pour réaliser la tâche de manière optimale. Il existe un bon nombre de problèmes qui semblent très faciles à résoudre pour les êtres humains mais dont la résolution via ordinateur est un mystère pour tout le monde.

   Exemple 1.2.2. *Parmi ces problèmes "faciles", la reconnaissance des personnes d'une même famille est un des plus illustratifs. Beaucoup de personnes sont capables de déterminer si deux personnes sont de la même famille en observant uniquement leur visage et ce, très facilement. Cependant, les meilleurs programmeurs au monde sont incapables de concevoir un agent capable de le faire sans apprentissage.*

   Il existe bien d'autres applications justifiant cette utilité comme le *data mining*, la *reconnaissance de caractères manuscrits* ou encore la *robotique*, ces applications sont traitées en détails dans la section 1.4

1.2.2 Fonctionnement

Le fonctionnement d'un agent apprenant dépend du *feedback* auquel il a accès. Le feedback représente les informations qui sont données/retournées à l'agent lorsqu'il commence à agir ou lors de son fonctionnement. Il s'agit là de ce qui va permettre à l'agent d'**apprendre**. Ce feedback peut être de trois types distincts. Ceux-ci définissant de cette manière trois types d'apprentissage (qui seront traités plus en profondeur dans la section 1.3) :

1. L'***apprentissage non-supervisé***.
   L'agent apprend de données en entrée bien qu'aucun feedback explicite ne soit fourni. Le plus connu de ces apprentissages non-supervisé est le *clustering*. Il consiste à détecter des clusters (des groupes de données) potentiellement utiles sur des exemples de données en entrée.

   Exemple 1.2.3. *Un taxi automatisé développe graduellement un concept de "jours de bon/mauvais trafic" sans même avoir eu d'exemple faisant état de l'existence de ce concept.*

2. L'***apprentissage supervisé***.
   L'agent observe quelques exemples de couples entrée/sortie et apprend une fonction qui permet de faire correspondre ces entrées aux sorties.

Exemple 1.2.4. *Un agent devant servir à prévoir des feux de forêts peut apprendre une fonction prenant plusieurs paramètres (comme le taux d'humidité, la température, ...) via des relevés de ces paramètres lors de réels feux de forêts et lorsqu'il n'y a pas eu de feu.*

L'agent présenté dans l'exemple 1.2.2 est un agent d'apprentissage supervisé. Il existe également de tels agents utilisés pour des prévisions boursières, météorologiques, des prévisions de prix de maison sur base de certains critères.

3. L'***apprentissage par renforcement***.
   L'agent apprend de séries de renforcements, récompenses ou punitions.

   Exemple 1.2.5. *Un agent du style Apple Genius servant, sur base d'une communauté, à proposer des musiques inconnues à l'utilisateur pourrait recevoir une récompense à chaque fois qu'un utilisateur achète une de ces musiques.*

Dans les deux premiers cas, il s'agit donc d'apprendre via des exemples (ou *données*) fournis préalablement en entrée. Un agent peut en effet tirer avantage d'exemples pour capturer des caractéristiques intéressantes de leur distribution (inconnue) de probabilité. Ils illustrent les relations entre des variables observées. Un axe majeur de la recherche dans l'apprentissage automatique est d'apprendre automatiquement à reconnaître des motifs complexes et prendre des décisions intelligentes en se basant sur ces exemples. La difficulté repose dans le fait que l'ensemble de tous les comportements donnés possibles est trop grand pour être couvert par l'ensemble des exemples observés. De par cette constatation, l'agent se doit de généraliser les exemples, de trouver une sorte de règle ou de loi qui pourrait s'y appliquer afin de l'utiliser sur de nouveaux cas et obtenir de bons résultats.

Sous et sur-évaluation. Au vu du précédent paragraphe, il est certain qu'un nombre trop minime d'exemples réduit grandement les possibilités de bons résultats pour l'agent. C'est ce qui est appelé *sous-évaluation*. Pour éviter cette dernière, il faut donc fournir un grand nombre d'exemples à l'agent afin qu'il puisse avoir assez d'informations pour tirer une loi ou une règle de ces données. Cependant, il ne faut pour autant pas en fournir "de trop" car l'agent pourrait alors avoir tendance à être trop précis (c'est-à-dire se coller entièrement aux exemples). Ceci porte le nom de *sur-évaluation*. L'agent souffrant de cette dernière n'est pas assez général et se montre peu apte à être appliqué sur de nouveaux cas. L'exemple ci-dessous permet d'illustrer ces deux notions.

Exemple 1.2.6. (Interpolation polynomiale)
Il s'agit d'un exemple très simple d'apprentissage supervisé. En effet, l'interpolation consiste à chercher une fonction mathématique passant par les points (x, y) *fournis. Le* x *pourrait par exemple représenter la superficie d'une maison et* y *son*

prix. L'idée est donc de trouver une fonction qui permet de prédire le prix de n'importe quelle maison si on connaît sa superficie. Pour y parvenir, l'agent se base sur des couples (superficie,prix) déjà connus. En considérant l'exemple présenté en figure 1.1, il apparaît qu'il existe plusieurs fonctions possibles que l'agent peut apprendre pour passer par les points fournis.

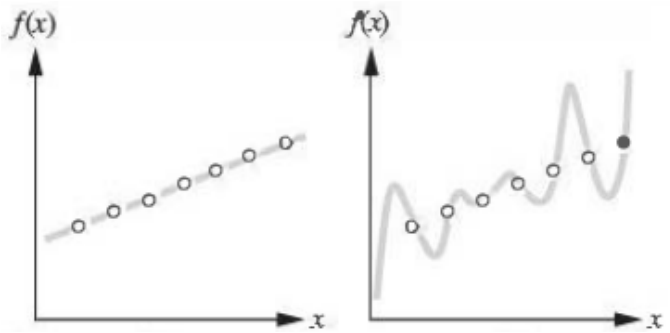
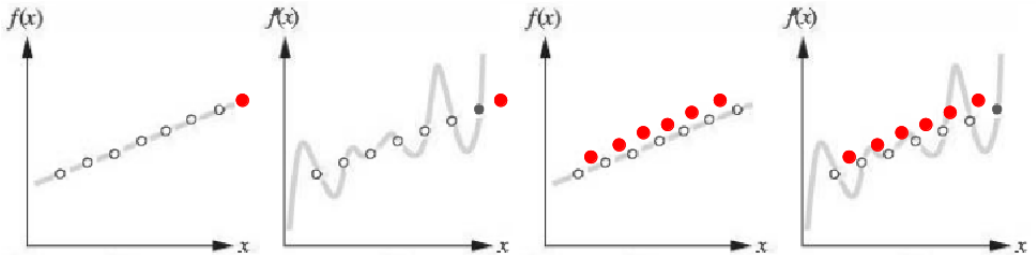


FIGURE 1.1 – Exemple d'interpolation polynomiale avec un polynôme de degré 1 et un polynôme de degré 6 [1]

Le modèle linéaire semble être le plus approprié car les chances pour que le polynôme de degré 6 soit capable d'englober des points non fournis sont très faibles. Comme montré dans la figure 1.2a, le simple ajout d'un point supplémentaire montre que le modèle polynomial de degré 6 ne se colle pas parfaitement à la situation réelle, il s'agit d'un cas de ***sur-évaluation****. Le modèle linéaire semble quant à lui parfaitement adéquat.*



(a) Sur-évaluation (b) Sous-évaluation

FIGURE 1.2 – Exemple de sur-évaluation et sous-évaluation

D'un autre côté, en jetant un regard à la figure 1.2b, en considérant que seuls les points blancs sont fournis à l'agent. Même si ce dernier choisit le modèle linéaire, lorsqu'il reçoit les points rouges, son modèle se trouve totalement erroné bien qu'il soit parfait pour les points auxquels il avait accès jusque-là. Il s'agit là d'un cas de ***sous-évaluation****.*

La figure 1.3 schématise le fonctionnement d'un agent apprenant de manière générale (l'agent ainsi représenté fait appel aux différents types d'apprentissage, ce qui n'est pas toujours le cas).

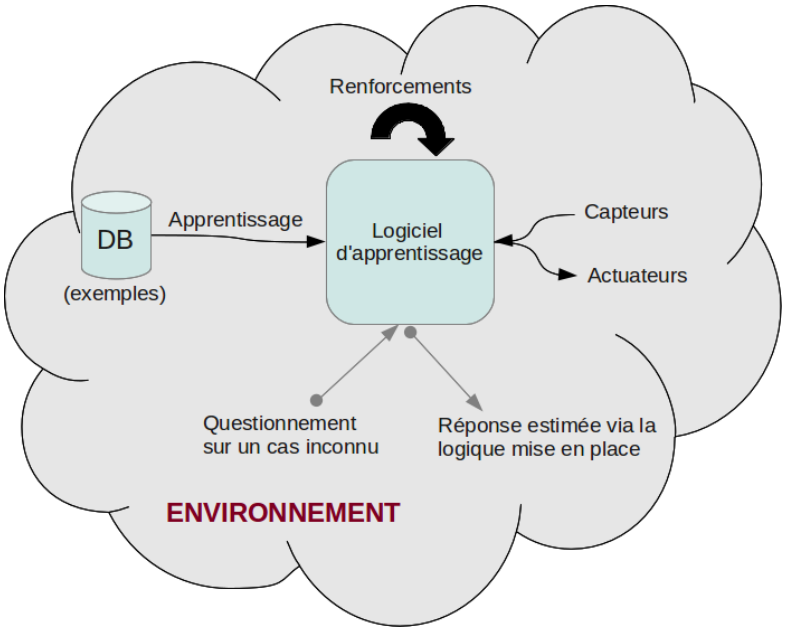


FIGURE 1.3 – Fonctionnement d'un agent apprenant

1.2.3 Performance, qualité et faisabilité

La faisabilité d'un apprentissage est une notion importante (à quoi bon développer un agent apprenant s'il est voué à l'échec ?) et la performance de celui-ci l'est tout autant. Le calcul de ces notions fait l'œuvre d'une branche particulière de l'apprentissage automatique connue sous le nom de "*théorie de l'apprentissage informatique*".

Performance et faisabilité. La théorie de l'apprentissage permet de fournir des bornes probabilistes sur la performance d'un agent apprenant. L'emploi du mot "performance" n'est pas innocente, ces calculs ne se basant que sur une étude empirique des résultats de l'apprentissage, il ne s'agit donc pas de la *qualité* de l'apprentissage. En outre, l'apprentissage est considéré *faisable* s'il peut être effectué en temps polynomial.

Qualité. La qualité de l'apprentissage et de l'analyse dépendent de beaucoup de paramètres et est souvent soumise à appréciation. Parmi ces paramètres on peut compter entre autres :

- **Le besoin** – par exemple, un agent estimant les futurs achats d'un client peut se permettre une plus grande marge d'erreur qu'un agent prévoyant une fission nucléaire.
- **Les données** – si les exemples fournis ne reflètent pas la réalité de manière fidèle, les résultats de l'agent ne seront pas très convaincants même si l'apprentissage utilisé est correct.
  Ces exemples dépendent de différents facteurs contraignants :
  - leur nombre : suffisamment pour éviter la sous-évaluation mais pas de trop pour éviter la sur-évaluation, un grand nombre pouvant aussi consumer la mémoire disponible de manière rapide ;
  - le nombre et la qualité des variables (attributs) utilisées : par exemple, n'utiliser qu'une seule variable pour prédire une fission nucléaire ne paraît pas très réaliste, de même utiliser une variable représentant le nombre d'insectes présents dans le laboratoire semble fort peu probant ;
  - le pourcentage de données manquantes : à savoir si les exemples sont assez représentatifs de l'entièreté de la réalité ou d'une bonne partie ;
  - le bruit : s'il y a beaucoup d'exemples isolés représentant des cas exceptionnels, ceux-ci peuvent empêcher l'apprentissage de dégager une loi/une règle des données initiales.
- **La complexité du modèle**, au final la qualité est surtout quantifiable sur la sortie de l'agent et donc sur la complexité du modèle. En effet, une série de nombres est beaucoup moins compréhensible qu'un graphique les mettant en place.

Exemple 1.2.7. *Pour en revenir à l'exemple de l'interpolation, les exemples (points) peuvent influencer l'agent de la manière suivante :*

- *Le nombre : il s'agit du nombre de points donnés, comme montré dans l'exemple 1.2.6, un trop grand nombre va impliquer que des polynômes de haut degré seront nécessaires et à l'image du polynôme de degré 6, il se peut qu'il s'éloigne de la réalité. A l'inverse, s'il n'y a pas assez de points, il est impossible de prédire le bon modèle.*
- *Le nombre et la qualité des variables : il se peut que la superficie d'une maison n'influence pas du tout son prix ou qu'elle ne soit pas suffisante pour en déterminer la valeur, la qualité de la variable "superficie" ainsi que le nombre de variables est donc mis à mal dans ce cas.*
- *Le pourcentage de données manquantes : cela rejoint le nombre, il se peut qu'il y ait assez d'exemples fournis mais que ceux-ci ne soient pas assez représentatifs de la distribution réelle.*
- *Le bruit : si un point (x_{out}, y_{out}) dont les coordonnées sont très éloignées des autres points figure dans l'ensemble des exemples fournis à l'agent, cela peut avoir comme effet que le polynôme calculé soit étriqué à cet*

endroit juste pour englober ce point. Un exemple d'un tel point pourrait représenter une maison ayant appartenu à une célébrité et dont le prix serait exceptionnellement élevé.

1.3 Les différents types d'apprentissage

La précédente section présentait la structure générale d'un agent apprenant ainsi que les différentes notions utilisées dans celui-ci. Elle avait également introduit de manière succincte les 3 principaux types d'apprentissage, ceux-ci différant fondamentalement par le feedback fourni à l'agent. La présente section, elle, décrit les différents types d'apprentissage en s'attardant particulièrement sur les trois principaux types cités au préalable.

1.3.1 L'apprentissage supervisé

Cette section présente les bases de l'apprentissage supervisé mais n'en fournit pas les détails, le lecteur peut se référer à AIMA[1] pour de plus amples informations. Celle-ci commence par définir formellement l'apprentissage pour ensuite en expliquer quelques principes de base. Elle se clôture par un exemple récapitulatif afin de permettre au lecteur de bien comprendre les notions développées au préalable.

Définition 1.4. *L'ensemble des exemples fournis à un agent apprenant afin qu'il en apprenne une loi générale est appelé* ***ensemble d'entraînement****.*

Définition 1.5. *Étant donné un ensemble d'entraînement contenant N exemples de paires entrée/sortie (x_1, y_1), (x_2, y_2), ... (x_N, y_N), où chaque y_j a été généré par une fonction inconnue $y = f(x)$,* ***la tâche de l'apprentissage supervisé*** *est de découvrir une fonction h qui approxime la vraie fonction f.*

Dans la définition formelle ci-dessus, les x et y peuvent représenter n'importe quelle valeur, ils ne sont pas obligatoirement des nombres.
L'apprentissage supervisé est dès lors aussi appelé :

- *classification* lorsque y appartient à un ensemble fini de valeur,
- *classification binaire* ou *booléenne* lorsqu'il n'y a que deux valeurs possibles pour y,
- *régression* lorsque y est un nombre.

La fonction h est appelée *fonction hypothèse*. Ainsi, l'apprentissage se résume à une recherche dans l'espace des hypothèses possibles d'une hypothèse qui "fonctionne bien" à la fois sur l'ensemble d'entraînement que sur de nouveaux exemples inconnus. Cet espace d'hypothèse peut être par exemple l'espace des fonctions polynomiales, des fonctions dérivables ou encore l'ensemble des réseaux bayésiens, des réseaux de neurones ou des arbres de décisions. Là où les

réseaux bayésiens permettent de calculer des probabilités conditionnelles, les arbres de décisions et les réseaux de neurones permettent de représenter n'importe quelle fonction booléenne (linéaire uniquement pour les arbres de décisions). Afin de calculer la précision d'une hypothèse, celle-ci est utilisée pour prédire la valeur y d'un ensemble test d'exemples. Une hypothèse est considérée comme "généralisant bien" dès lors qu'elle prédit correctement la valeur y pour de nouveaux exemples. Généralement, l'apprentissage ne s'attarde qu'aux hypothèses *consistantes*, celles-ci étant les seules hypothèses intéressantes.

Définition 1.6. *Une fonction hypothèse est dite* ***consistante*** *si elle s'accorde avec toutes les données. C'est-à-dire que pour toutes les valeurs de x des exemples de l'ensemble d'entraînement, elle fournit la bonne valeur de y.*

Selon l'espace choisi pour les hypothèses, il se peut que plusieurs hypothèses consistantes soient possibles. L'agent a dès lors besoin d'un moyen de choisir parmi elles celle qu'il doit préférer. Pour ce faire, le principe le plus utilisé est *le rasoir d'Ockham* qui dit sommairement qu'il faut privilégier l'hypothèse **consistante** la *plus simple*.

Le choix de l'espace des hypothèses. Il est crucial. En effet, il faut s'assurer que la fonction réelle (celle qui correspond à la réalité) figure dans cet espace. Si ce n'est pas le cas, les chances pour que l'agent trouve cette fonction sont évidemment nulles. Dès lors, un problème d'apprentissage est *réalisable* uniquement si l'espace des hypothèses contient la fonction réelle. De manière évidente, et malheureusement, il n'est pas toujours possible de savoir si un problème d'apprentissage est réalisable pour la simple et bonne raison que la fonction réelle est inconnue. Une solution à cela serait de prendre l'espace des hypothèses le plus large possible mais cela aurait un impact sur les performances de l'agent. Certains algorithmes utilisent tout de même un ensemble expressément large mais l'agrémentent de probabilités afin de privilégier les hypothèses les plus simples.

Exemple 1.3.1. (Interpolation polynomiale)
En considérant l'exemple précédemment utilisé et représenté en figure 1.4 et en limitant l'ensemble des hypothèses à l'ensemble des polynômes, il apparaît qu'il existe déjà au moins deux hypothèses consistantes :

- *un polynôme de degré* 6*,*
- *une droite (polynôme de degré* 1*)*

Dans cette situation, le principe du ***rasoir d'Ockham*** *indique que la meilleure hypothèse est l'hypothèse linéaire.*

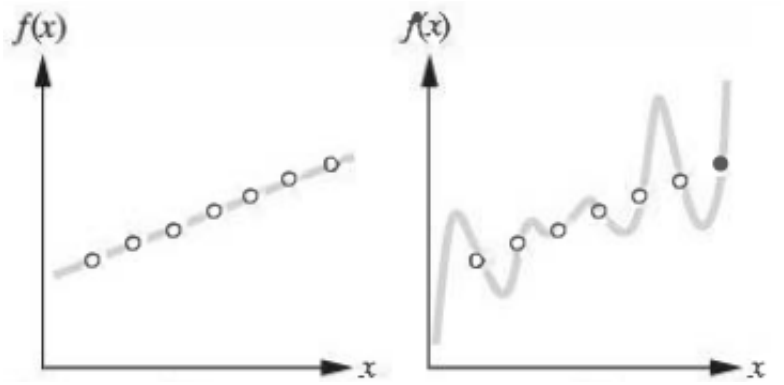


FIGURE 1.4 – Exemple d'interpolation polynomiale [1]

1.3.2 L'apprentissage non-supervisé

Ce type d'apprentissage est très proche de l'apprentissage supervisé. En effet, la différence fondamentale vient du fait qu'il n'y aucun feedback fourni à l'agent. C'est-à-dire que les exemples fournis à l'agent n'ont pas de valeur y associée.

Définition 1.7. *Soit un ensemble $X = (x_i)\ \forall i \in \{1, ..., n\}$ d'exemples (ou points), l'**apprentissage non-supervisé** a comme but de trouver une structure dans ces données X mettant en évidence certaines (dis)similarités entre celles-ci.*

Ils existent plusieurs techniques d'apprentissage non-supervisé telles que l'*estimation des quantiles*, le *clustering*, la *détection des outliers* et la *réduction de la dimensionalité*. La plus intéressante de ces techniques est le **clustering** et c'est cette technique qui est exploitée ici. Pour plus d'informations sur les autres techniques et sur l'apprentissage non-supervisé en général, le lecteur peut se référer aux ouvrages qui ont été consultés pour la rédaction de cette section [4, 5].

Clustering

Le clustering a pour but d'organiser une collection de données, d'exemples, de points dans des *clusters* (des ensembles) de manière à ce que l'affirmation suivante soit vérifiée :

*Des points d'un même cluster sont plus **similaires**/proches les uns des autres que des points appartenant à des clusters différents.*

Cette notion de *similarité* peut être exprimée de beaucoup de manières différentes, celle-ci dépendant généralement du sujet de l'étude, des hypothèses spécifiques au domaine et de la connaissance initiale du problème. Le clustering est généralement utilisé lorsqu'aucune information n'est disponible en ce qui concerne l'appartenance des données à une classe prédéfinie. Pour cette raison, le clustering est traditionnellement vu comme une partie de l'apprentissage non-supervisé.

Les différents types de clustering. Ces différents types peuvent se distinguer selon plusieurs critères :

1. **Objectif du clustering**
   Beaucoup de méthodes ont pour but de trouver une seule partition de la collection des objets en clusters, il s'agit de clustering *partitionnel*. Cependant, d'autres méthodes, dites de clustering *hiérarchique*, visent à créer une hiérarchie de clusters, celle-ci pouvant fournir plus de flexibilité. Une partition des données pouvant être obtenue d'une hiérarchie en coupant cette dernière à un certain niveau désiré. Ces deux types de clustering ainsi définis définissent deux grandes catégories de clustering.
2. **Nature des données**
   La plupart des méthodes de clustering ont été développées pour des données numériques, mais certaines peuvent utiliser des données catégoriques ou un mix de données catégoriques et numériques.
3. **Nature de l'information disponible**
   Beaucoup de méthodes se basent sur des représentations riches des données que l'on définit comme des prototypes, distributions de donnée, intervalles multidimensionnels,... à coté de calculs de (di)similarités. D'autres méthodes nécessitent uniquement l'évaluation des (di)similarités entre données deux à deux ; en imposant moins de restrictions sur les données, ces méthodes ont généralement une complexité de calcul supérieure.
4. **Nature des clusters**
   Les clusters peuvent être *flous* ou *stricts*. La différence entre ces deux types de clusters réside dans la fonction déterminant le *degré d'appartenance* d'une donnée à un cluster.
   Soit X l'ensemble des données, soit C l'ensemble des clusters, cette fonction est définie comme suit (m pour *membership*) :

   $$m : X \times C \mapsto D.$$

   La différence citée plus haut concerne l'ensemble D :
   – dans les clusters flous, $D = [0, 1]$;
   – dans les clusters stricts, $D = \{0, 1\}$, c'est-à-dire que soit la donnée appartient au cluster, soit elle ne lui appartient pas.

   L'avantage majeur des clusters flous est qu'ils permettent une approche plus générale des problèmes de clustering. En effet, ceux-ci permettent aux données d'appartenir à plusieurs clusters même si ceux-ci n'ont aucun lien hiérarchique. Ceci pouvant servir à modéliser les clustering où certains clusters se chevauchent.
   En outre, il est toujours possible d'obtenir des clusters stricts à partir de clusters flous. De manière évidente et simple, pour trouver m_{STRICT} la fonction d'appartenance des clusters stricts, il suffit de définir une certaine borne $b \in [0, 1]$ telle que $\forall$ donnée $d \in X$ et cluster $c \in C$:

– si $m(d, c) \geq b$, alors $m_{\text{STRICT}}(d, c) = 1$,
– si $m(d, c) < b$, alors $m_{\text{STRICT}}(d, c) = 0$,

Il existe d'autres critères plus poussés comme la distinction entre ensembles denses et ensembles compacts mais ils ne sont pas développés ici, bien qu'ils soient discutés dans plusieurs articles [5].

Clustering partitionnel. Cette catégorie de clustering, comme expliqué plus haut, a pour but d'obtenir directement une seule partition d'une collection de données dans des clusters. Elle regroupe une multitude de méthodes permettant de fournir cette unique partition. Une grande partie de ces méthodes est basée sur une optimisation itérative d'une fonction *"critère"* reflétant l'"accord" entre les données et la partition. La différence entre les différentes méthodes peut donc se faire sur la fonction critère à optimiser (et/ou sur la fonction de dissimilarité sur laquelle elle repose) ainsi que la manière de l'optimiser.

1. **Méthodes utilisant l'erreur quadratique comme fonction critère**.
   Ces méthodes se basent toutes sur la possibilité de représenter chaque cluster par un *prototype*. Ce prototype est une entité unique représentant un cluster, il peut s'agir d'une donnée fictive représentant la moyenne des données d'un cluster (*centroïde*, comme pour l'algorithme bien connu **"k-means"** [6]) ou de la donnée dont la similarité moyenne avec toutes les données du cluster est la plus grande (*médoïde*, approche **"k-medoids"** [7]), ou d'autres encore. Comme introduit par le titre, ces méthodes utilisent l'erreur quadratique comme fonction critère. Cette erreur est définie comme suit :
   Soient :
   – $C = (c_j)_{j \in \{1,\dots,m\}}$ l'ensemble des clusters,
   – $C_j = (y_k)_{k \in \{1,\dots,|C_j|\}}$ l'ensemble des données appartenant au cluster c_j,
   – $C^p = (c_j^p)_{j \in \{1,\dots,m\}}$ l'ensemble des prototypes de clusters tel que c_j^p soit le prototype du cluster c_j.

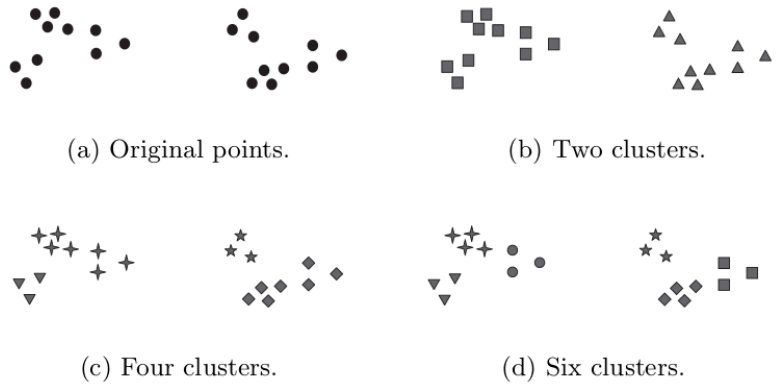
   L'erreur quadratique se définit alors comme la somme des carrés des dissimilarités entre chaque donnée et le prototype du cluster auquel la donnée est affectée, ou, de manière mathématique :

   $$e = \sum_{c_j \in C} \sum_{y_k \in C_j} dissim(c_j^p, y_k)^2$$

   Les méthodes décrites ci-dessus utilisent des clusters stricts, mais il existe des versions de celles-ci adaptées pour les clusters flous. L'avantage de ces méthodes est qu'elles sont plus aptes à éviter les minima locaux de la fonction de coût et, comme déjà dit plus haut, qu'elles peuvent modéliser des situations où les clusters se chevauchent. Parmi ces méthodes, la plus connue est sans doute **Fuzzy C-Means** [8].

   Beaucoup de méthodes supposent que le nombre de clusters était connu avant le clustering : étant donné que c'est rarement le cas, des techniques

pour trouver un nombre "approprié" de clusters ont été mises au point. Il s'agit d'un problème important pour le clustering partitionnel en général, comme le montre la figure 1.5. Pour les méthodes basées sur l'erreur au carré, le problème est résolu en partie en ajoutant un terme de régularisation à la fonction de coût. C'est le cas par exemple pour la méthode d'agglomération compétitive [9] où les clusters se "battent" pour recruter une donnée et le nombre de clusters est progressivement réduit jusqu'à ce qu'un optimum soit atteint. Avec de telles solutions, au lieu de contrôler le nombre de clusters, il faut contrôler un terme de régularisation, ce qui est souvent plus pratique. Une autre solution est d'utiliser un indice de validité de cluster pour sélectionner à posteriori le nombre approprié de clusters.



(a) Original points. (b) Two clusters.

(c) Four clusters. (d) Six clusters.

FIGURE 1.5 – Plusieurs clusterings à partir d'un même ensemble d'exemples (image tirée de [10])

2. **Les méthodes se basant sur la densité**
   Ces méthodes considèrent que les clusters sont des ensembles denses de données séparés par des régions moins denses ; les clusters pouvant avoir une forme arbitraire et les données distribuées arbitrairement. Beaucoup de ces méthodes, comme **DBSCAN** [11], se basent sur une étude de la densité des données dans le voisinage de chaque donnée. D'autres utilisent la théorie des graphes, les données sont alors représentées comme des nœuds dans un graphe et la dissimilarité entre deux données est le poids de l'arête les reliant. Dans la plupart de ces méthodes un cluster est un sous-graphe qui reste connecté après le retrait des arêtes de plus gros poids [12]. D'autres méthodes se basent sur un quadrillage de l'espace comme **DenClue** [13] ou encore sur l'extraction de cliques d'un graphe.
3. **Les méthodes mélange-résolution**
   Ces méthodes supposent que les données dans un cluster sont dessinées via une certaine distribution (Gaussienne souvent) et essaient d'estimer les paramètres de ces distributions. L'introduction du célèbre algorithme "Expectation Maximization" (**EM**) [14] fut un pas important dans la résolution

de tels problèmes d'estimation.
Ces méthodes font des suppositions assez fortes en ce qui concerne la distribution des données. La plupart de ces méthodes voient chaque cluster comme une simple distribution et contraignent donc fortement les formes des clusters ; ceci explique pourquoi ces méthodes sont considérées à part (elles auraient pu être considérées comme des méthodes se basant sur la densité).

Clustering hiérarchique. Le but d'un tel clustering est d'obtenir une hiérarchie de clusters, appellée dendrogramme, qui montre comment les clusters sont reliés entre eux. Ces méthodes procèdent soit itérativement en fusionnant des petits clusters en de plus gros (*algorithmes agglomératifs*, de loin les plus communs) ou en coupant des gros clusters en plus petits (*algorithmes divisants*). Une partition des données peut être obtenue en coupant le dendrogramme à un niveau désiré.

Les algorithmes agglomératifs ont besoin de critères pour fusionner les petits clusters dans des plus gros. La plupart des critères concerne la fusion d'une paire de clusters (créant des arbres binaires) et sont des variantes de critères classiques :

- *single-link* [15], critère selon lequel la dissimilarité entre les clusters C_1 et C_2 est donnée par la similarité entre les deux éléments (appartenant chacun à un cluster différent) les plus proches, soit :

$$\min_{x \in C_1, y \in C_2} dissim(x, y) \text{ ou } \max_{x \in C_1, y \in C_2} sim(x, y)$$

- *complete-link* [16], critère selon lequel la dissimilarité entre les clusters C_1 et C_2 est donnée par la similarité entre les deux éléments (appartenant chacun à un cluster différent) les plus éloignés, soit :

$$\min_{x \in C_1, y \in C_2} sim(x, y) \text{ ou } \max_{x \in C_1, y \in C_2} dissim(x, y)$$

- *minimum-variance* [17], critère selon lequel la dissimilarité entre les clusters C_1 et C_2 est calculé sur base des variances des distributions des données dans ces deux clusters ;
- *group-average* [10], critère selon lequel la dissimilarité entre les clusters C_1 et C_2 est donnée par la moyenne des dissimilarités entre paires d'éléments mixtes (un élément dans chaque cluster),soit

$$\frac{1}{|C_1||C_2|} \sum_{x \in C_1, y \in C_2} dissim(x, y) \text{ ou } \frac{1}{|C_1||C_2|} \sum_{x \in C_1, y \in C_2} sim(x, y)$$

L'utilisation du critère single-link peut être mis en relation avec les méthodes basées sur la densité mais produit souvent des effets contrariants : les clusters

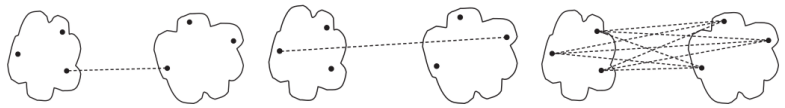


FIGURE 1.6 – Single, complete-link et group-average (distance euclidienne) (image tirée de [10])

qui sont liés par une "ligne" de données (cf figure 1.7) ne peuvent pas être séparés ou la plupart des données sont individuellement fusionnées à un cluster (ou quelques uns). Les deux autres critères sont plus proches des méthodes de l'erreur quadratique.

FIGURE 1.7 – Exemple de clusters reliés par une "ligne" de données.

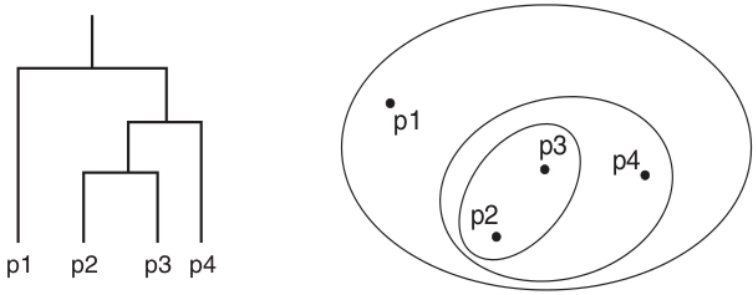


FIGURE 1.8 – Exemple d'un dendrogramme et des clusters associés (tirés de [10])

1.3.3 L'apprentissage semi-supervisé

Cette section se base sur l'ouvrage de Chapelle, Scholkopf et Zien [18] qui est un ouvrage de référence pour ce type d'apprentissage. La présente section se contente d'établir les bases et de donner les intuitions concernant l'apprentissage semi-supervisé mais ne rentre pas dans les détails.

Dans la réalité, la frontière entre l'apprentissage supervisé et l'apprentissage non-supervisé n'est pas aussi discriminante. L'apprentissage semi-supervisé en est la preuve. Ce type d'apprentissage est à mi-chemin entre ces deux types d'apprentissage : en plus d'exemples dont la valeur y est inconnue, l'agent reçoit de l'information supervisée mais pas forcément pour tous les exemples.

L'information supervisée supplémentaire. Celle-ci peut être de plusieurs formes. Souvent, il s'agit de la valeur y de certains exemples, dans ce cas l'ensemble des exemples $X = (x_1, ..., x_n)$ peut être partitionné comme suit :

- $X_l := (x_1, ..., x_l)$ avec les valeurs y associées $Y_l := (y_1, ..., y_l)$,
- $X_u := (x_{l+1}, ..., x_{l+u})$ dont les valeurs y sont inconnues.

Il s'agit de la formulation standard de l'apprentissage semi-supervisé. Cependant d'autres formes de supervision "partielle" sont également possibles. Par exemple, l'apprentissage peut intégrer des **contraintes** comme "ces deux points doivent avoir la même valeur y". Il est également envisageable de manière plus générale que l'apprentissage soit aux prises avec des points dont la valeur y est inconnue suivant une certaine distribution. Dans ce cas la partie supervisée serait alors des informations supplémentaires renseignant l'agent quant à la distribution suivie par ces points.

1.3.4 L'apprentissage par renforcement

Cette section introduit le type d'apprentissage qui est le plus utilisé dans ce mémoire : l'apprentissage par renforcement. Elle commence par une présentation et une définition concise avant d'expliquer en détails chaque élément du système d'apprentissage par renforcement. Pour mener à bien cette tâche, cette section se base sur le livre faisant office de référence sur le sujet : RLAI écrit par Sutton et Barto [19].

Présentation et définition

Définition 1.8. *L'apprentissage par renforcement consiste à apprendre quelle action effectuer selon la situation dans le but de maximiser un gain numérique significatif (idée de récompense). L'agent suivant ce type d'apprentissage n'est pas au courant des actions à effectuer, il doit les apprendre de lui-même en les essayant dans l'optique de trouver celle le récompensant le plus. Dans les agents les plus complexes (et les plus intéressants) les actions n'affectent pas seulement la récompense directe mais également les récompenses futures. Ces deux caractéristiques, la recherche par essai-erreur ainsi que la récompense retardée, sont ce qui définissent le plus l'apprentissage par renforcement.*

Un apprentissage par renforcement n'est pas défini par des méthodes d'apprentissage mais en caractérisant un *problème d'apprentissage*. L'idée de base est de capturer simplement les aspects les plus importants du réel problème d'apprentissage en se mettant à la place d'un agent d'apprentissage interagissant avec son environnement pour parvenir à un but. L'agent doit forcément pouvoir, comme tout agent :

- sentir l'état de l'environnement (via ses **senseurs**)
- agir sur l'environnement (via ses **actuateurs**)
- posséder un ou plusieurs but(s) .

Il s'agit là des trois éléments que doivent contenir la formulation.

Un des problèmes les plus ambitieux et spécifiques à l'apprentissage par renforcement est de trouver la bonne balance entre "apprendre" et donc tester de nouvelles actions et "agir de manière optimale". En effet, pour obtenir une meilleure récompense, l'agent va tenter de faire appel aux actions qu'il a utilisées par le passé qui ont rapporté le plus. Mais pour découvrir de telles actions, il lui faut essayer de nouvelles actions. Le dilemme est réel, car ni le fait d'essayer tout le temps de nouvelles actions ni le fait de répéter les mêmes actions perpétuellement n'est optimal. Il convient donc de faire tester à l'agent une variété d'actions et de progressivement choisir celles qui apparaissent être les meilleures.

Les éléments d'un apprentissage par renforcement

Mis à part l'agent et l'environnement, il y a principalement quatre éléments qui composent un système d'apprentissage par renforcement (en plus des actuateurs et des senseurs) :

1. la *politique*,
2. la *fonction de récompense*,
3. la *fonction de valeur*,
4. un *modèle de l'environnement* (optionnellement).

La fonction de récompense. La fonction de récompense définit le but du problème d'apprentissage par renforcement. De manière plus rigoureuse il s'agit d'un mapping entre chaque état perçu (ou une paire état-action) et une valeur numérique représentant la désirabilité de cet état. Le seul et unique but de l'agent est de maximiser la récompense totale obtenue sur le long terme.
Cette fonction caractérise le fait que les actions de l'agent sont *bonnes ou mauvaises*. On peut y voir une analogie avec le "plaisir" et la "douleur" pour un système biologique. Il est très important que cette fonction ne soit pas modifiable par l'agent, sinon celui-ci pourrait s'attribuer des récompenses arbitrairement élevées pour n'importe quelle action. Par contre elle peut servir de base pour modifier la politique suivie par l'agent. En effet, si une action suivie par la politique amène une sanction à l'agent alors qu'il y a une action qui lui permet de récupérer une récompense, il est sans doute judicieux de privilégier l'action procurant une récompense.

Exemple 1.3.2 (inspiré partiellement de [1]). *Un robot aspirateur évolue dans un environnement divisé en* 16 *cases et il est considéré qu'il commence son exploration en (1,1). L'environnement est illustré dans la figure 1.9.*

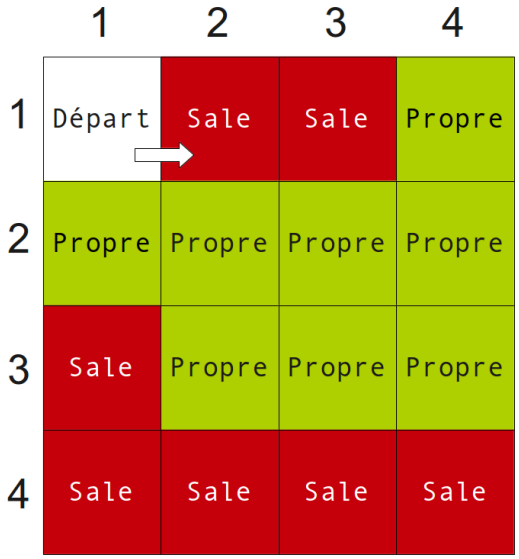


FIGURE 1.9 – Environnement du robot aspirateur

Sa fonction de récompense est simple : +1 quand il aspire un carré sale (on considère qu'il aspire dès qu'il se déplace sur la case et que la case devient propre et le reste par la suite). En considérant uniquement cette fonction de récompense, les choix se proposant à l'agent sont :

- *aller en (1,2) "Sale"* $\rightarrow f_{\text{REWARD}}((1,2)) = +1$,
- *aller en (2,1) "Propre"* $\rightarrow f_{\text{REWARD}}((2,1)) = 0$.

L'agent va donc privilégier le déplacement en $(1,2)$ *alors que dans le futur il va s'enfermer dans une région entièrement entourée de "Propre" et qu'il aurait été plus sensé d'aller en* $(2,1)$ *pour s'orienter des* 5 *carrés sales situés dans le bas de l'environnement. C'est ce qui motive l'existence de la fonction de valeur, décrite dans le paragraphe ci-dessous.*

La fonction de valeur. La fonction de valeur est utilisée pour pallier à un problème de la fonction de récompense : elle ne se projette pas dans le long terme. En effet cette dernière ne permet de prendre que des décisions localement optimales. La fonction de valeur va associer à chaque état une valeur représentant la récompense totale ou moyenne à laquelle l'agent peut s'attendre dans le futur en choisissant l'action menant à cet état. Cette fonction prend donc en compte les états qui sont supposés suivre l'état considéré et leur valeur de désirabilité. Elle représente donc une désirabilité à long terme. Définie de cette façon, il est donc très facilement envisageable qu'un état ait une valeur de désirabilité élevée

(grande récompense) mais une valeur très pauvre en terme de fonction de valeur (il suffit qu'il soit suivi par des états de très mauvaise désirabilité) ; et vice-versa.

Les récompenses sont primaires tandis que les valeurs sont secondaires. En effet, s'il n'y avait pas de récompenses, il n'y aurait pas de valeurs. Cependant, les décisions se feront pratiquement entièrement sur les valeurs des états vu que le but ultime est de maximiser la somme de récompenses à long terme. Le plus gros problème réside dans l'évaluation de celles-ci. Effectivement, elles doivent être estimées et ré-estimées sur base des séquences d'observations faites par l'agent. Effectuer ces calculs de manière efficace est une des parties les plus importantes de la conception d'un agent d'apprentissage par renforcement.

Exemple 1.3.3. *Reprenons ce robot aspirateur et son environnement :*

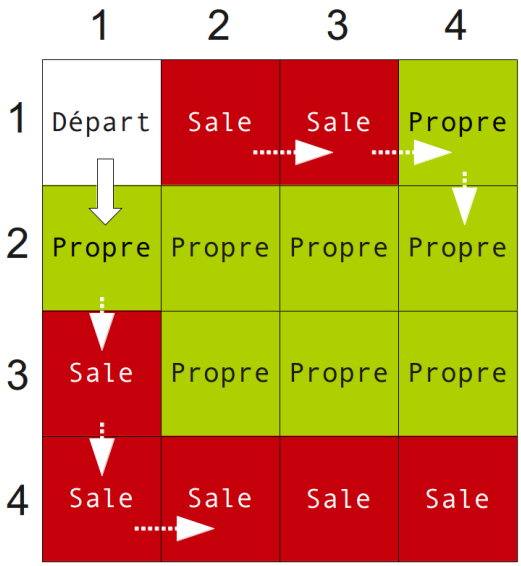


FIGURE 1.10 – Environnement du robot aspirateur

La fonction de valeur peut être définie comme la somme des récompenses possibles sur le meilleur chemin choisi à partir de l'état désiré (avec une profondeur maximale de quatre par exemple). A nouveau les choix se proposant à l'agent sont :

- *aller en (1,2) "Sale"* $\rightarrow f_{\text{REWARD}}((1,2)) = +1$ *mais* $f_{\text{VALUE}}((1,2)) = 2$
- *aller en (2,1) "Propre"* $\rightarrow f_{\text{REWARD}}((2,1)) = 0$ *mais* $f_{\text{VALUE}}((2,1)) = 3$

Les chemins permettant d'obtenir les valeurs de la fonction de valeur sont indiqués sur la figure 1.10. L'agent va donc privilégier la meilleure action au point de vue valeur et se déplacer en $(2,1)$ *qui est le meilleur choix possible.*

La fonction de valeur n'est cependant pas indispensable, d'autres méthodes comme le "simulated annealing", les algorithmes génétiques ou encore la programmation génétique, cherchent directement dans l'espace des politiques sans même faire appel aux fonctions de valeur. Ces méthodes sont appelées "évolutionnaires" car elles fonctionnent de manière analogue à l'évolution biologique créant des organismes avec des comportements compétents même si ces organismes n'apprennent pas lors de leur temps de vie. Ces méthodes évolutionnaires possèdent un avantage sur les problèmes sur lesquels l'agent d'apprentissage ne peut sentir l'état de l'environnement de manière précise. Elles ont cependant un inconvénient, elles n'apprennent pas de leurs interactions avec l'environnement.

Le modèle de l'environnement. Le modèle imite l'environnement. Par exemple, pour un état et une action donnée, le modèle pourrait prédire l'état suivant résultant de cette combinaison et donc la future récompense. Le modèle est utilisé pour faire de la *planification*. Cette planification désigne toute sorte de décision sur un plan d'action en considérant les futures situations possibles avant qu'elles n'aient été réellement expérimentées. Cette manière de procéder (utiliser des modèles pour l'apprentissage par renforcement) est assez récente, auparavant les agents n'utilisaient que la simple technique "test et erreur". Les méthodes actuelles d'apprentissage par renforcement couvrent le spectre du bas-niveau (test et erreur) au haut-niveau (la *planification délibérative*). Cette dernière permet à l'agent de prendre tout le temps nécessaire pour évaluer les différents choix possibles (et faire des simulations si nécessaire) afin de faire le choix optimal. Elle est opposée à la *planification réactive* qui contraint l'agent à réagir dans un temps imparti, celui-ci n'a donc pas assez de temps pour évaluer tous les chemins qui s'offrent à lui de fond en comble.

La politique. De manière formelle, soit S l'ensemble des états et A l'ensemble des actions, la politique est la fonction π telle que :

$$\begin{aligned} \pi : \quad S \times A \quad &\rightarrow [0,1] \\ (s,a) \quad &\rightarrow \pi_t(s,a) \end{aligned}$$

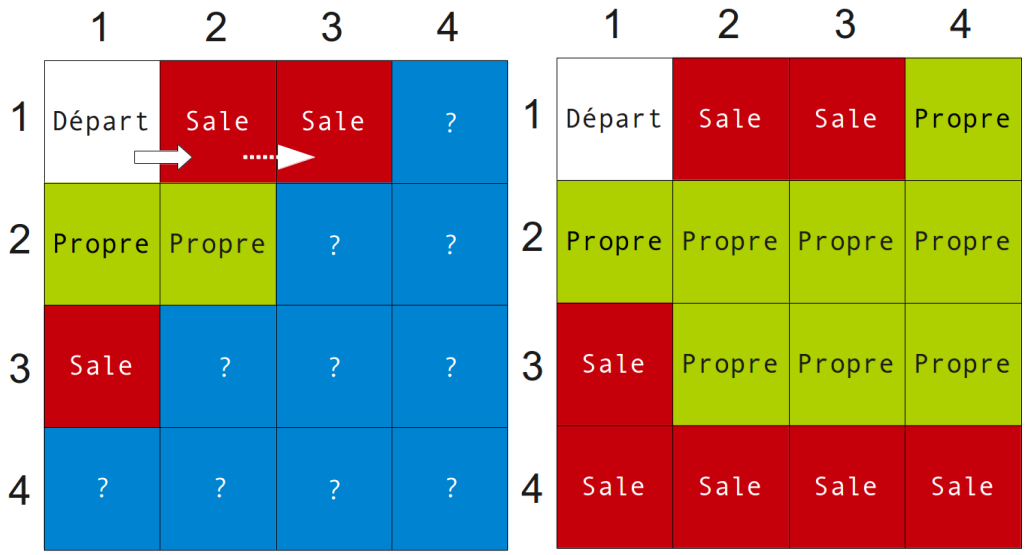
$\pi_t(s,a)$ représentant la probabilité pour que $s_t = s$ et $a_t = a$, soit que l'action choisie par l'agent et l'état dans lequel l'environnement se trouve à l'instant t soient a et s.

La politique définit le comportement de l'agent d'apprentissage à un moment donné. Comme déjà expliqué plus tôt, il s'agit d'une fonction faisant correspondre les perceptions faites de l'état de l'environnement et les actions à faire lorsque l'environnement est dans l'un de ces états. Il s'agit de l'élément le plus important de l'agent car il est suffisant pour déterminer son comportement.

Il paraît évident qu'une politique *gloutonne*, c'est-à-dire ne se basant que sur les récompenses et effectuant donc des choix localement optimaux, n'atteindra pas le but de maximiser la somme totale des récompenses obtenues.

Afin de remédier à cela, il faut donc permettre à l'agent de choisir des actions qui ne sont pas forcément les meilleures sur le moment. Ceci est rendu possible via l'utilisation de la fonction de valeur afin de se projeter dans le futur. De manière plus générale, la politique doit s'attacher à trouver le bon équilibre entre *exploration* et *exploitation des connaissances*. L'exploration consiste à sélectionner des actions qui ne sont pas spécialement connues pour amener beaucoup de récompenses (de manière directe comme de manière future) afin d'explorer d'autres actions et ainsi s'échapper d'un éventuel optimum local. L'exploitation des connaissances consiste quant à elle à effectuer le meilleur choix possible. Cet équilibre est important dans les environnements qui ne sont pas totalement observables, c'est-à-dire que l'agent ne sait pas toujours ce qui se passera une fois qu'il aura fait une action. Il se peut que cette action soit la pire imaginable mais il se peut également qu'elle soit meilleure (ou qu'elle amène à une meilleure) que celle qui est considérée comme optimale à cet instant.

Exemple 1.3.4. *Toujours ce robot aspirateur. L'exemple qui suit vise à donner une intuition de la nécessité pour l'agent d'explorer un minimum son environnement. Imaginons à présent que l'agent robot aspirateur possède des capteurs ne lui permettant de voir que jusqu'à deux cases plus loin que sa position actuelle. Son environnement est donc limité aux cases assez proches de lui, comme décrit par la figure 1.11a.*



(a) Environnement vu par l'agent (b) Environnement réel

FIGURE 1.11 – Environnement du robot aspirateur

Comme vu dans l'exemple 1.3.3, la meilleure action est d'aller en $(2, 1)$. *Cependant, de la manière dont voit l'agent, sa fonction de valeur ne peut plus se projeter que deux étapes en avant et donc :*

– *aller en (1,2) "Sale"* $\rightarrow f_{\text{REWARD}}((1, 2)) = +1$ *mais* $f_{\text{VALUE}}((1, 2)) = 2$

– *aller en (2,1) "Propre"* $\rightarrow f_{\text{REWARD}}((2, 1)) = 0$ *mais* $f_{\text{VALUE}}((2, 1)) = 1$

S'il choisit la solution optimale à chaque fois, il ne va jamais aller vers le bas de l'environnement et donc jamais découvrir la rangée de case "Sale". C'est là le but de l'exploration, l'agent va à un moment décider d'explorer et donc de ne pas choisir l'action correspondant à la meilleure valeur mais une autre (généralement déterminée aléatoirement) afin d'explorer les actions et états peu ou pas utilisés.

La politique la plus simple permettant un équilibre entre exploration et exploitation s'appelle ϵ-*greedy*. Elle consiste simplement à :

– choisir, avec une probabilité de ϵ, une action au hasard afin d'explorer,

– agir de manière gloutonne dans tous les autres cas (probabilité de $1 - \epsilon$).

Le souci avec cette approche est que le choix aléatoire est fait de manière équiprobable sur toutes les actions disponibles, ce qui peut amener la politique à essayer autant de fois des mauvaises actions que des bonnes. Ainsi, quelques politiques ont été développées pour remédier à ce problème. Parmi ces politiques, la plus connue est *softmax*. En définissant $Q_t(a)$ comme la valeur estimée par l'agent de l'action a à l'instant t, softmax définit la probabilité de choisir l'action a comme :

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b \text{ action}} e^{\frac{Q_t(b)}{\tau}}}$$

Avec $\tau > 0$ qui est un paramètre positif appelé *température*. Plus la température est haute, plus les actions tendent à être équiprobables. Cette formule est celle d'une distribution de Boltzmann (ou Gibbs) et permet d'utiliser des valeurs arbitraires pour les récompenses et valeurs et ramener ces valeurs à l'intervalle $[0, 1]$ afin d'obtenir des probabilités.

La figure 1.12 résume schématiquement le fonctionnement d'un agent apprenant par renforcement ($r =$ reward, récompense – $a =$action – $s =$ state, état).

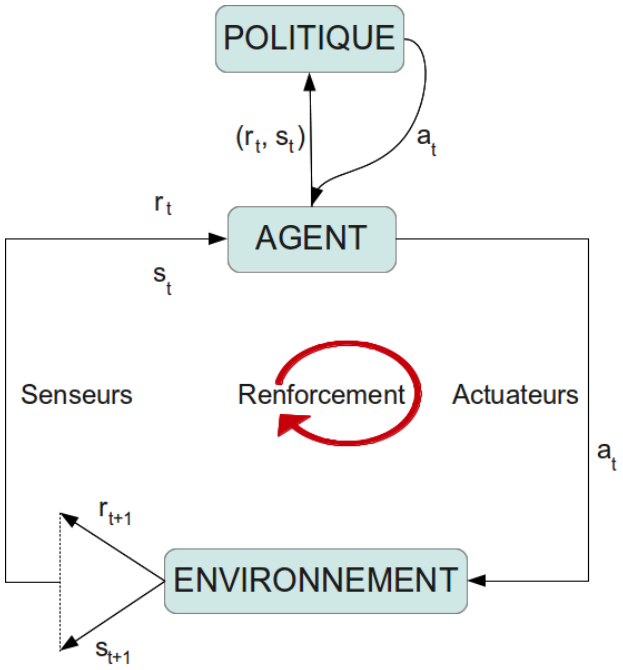


FIGURE 1.12 – Schéma du fonctionnement d'un agent apprenant par renforcement

1.4 Applications

Les applications d'apprentissage automatique sont nombreuses et variées. Ces applications peuvent être purement théoriques comme les régressions, très utilisées dans beaucoup de domaines utilisant de près ou de loin les statistiques. Elles peuvent également avoir des impacts plus directs sur la vie réelle. Parmi ces dernières, on trouve le data mining, la reconnaissance de caractères manuscrits ou encore la robotique. Cette section développe les applications sus-citées en donnant les intuitions et quelques détails. Elle n'a nullement la prétention d'être exhaustive, une simple visite sur un moteur de recherche permet d'en trouver toute une multitude.

1.4.1 Data mining

Le data mining est une discipline visant à trouver des motifs communs entre des données. Il a également pour but de résoudre des problèmes en analysant des données déjà présentes en base de donnée. Un exemple bien connu de l'application du data mining est **le problème de la météo**. Il s'agit d'un problème simpliste et fictif permettant de montrer la puissance du data mining, notamment au travers de règles de classifications. Ces règles, comme leur nom l'indique sont obtenues via une **classification**, ou, autrement dit, un **apprentissage supervisé** [10, 20].

Le problème consiste à considérer plusieurs variables renseignant sur l'état de la météo afin de prendre la décision de sortir pour jouer ou de rester à l'intérieur. Ces variables sont en général l'apparence du ciel, la température, l'humidité, le fait qu'il y ait du vent et donc le fait qu'il faille sortir ou non. Le modèle le plus simple utilisable pour modéliser cette classification est l'arbre de décision, celui-ci pouvant être comme illustré en figure 1.13 (cela dépend bien évidemment des exemples fournis).

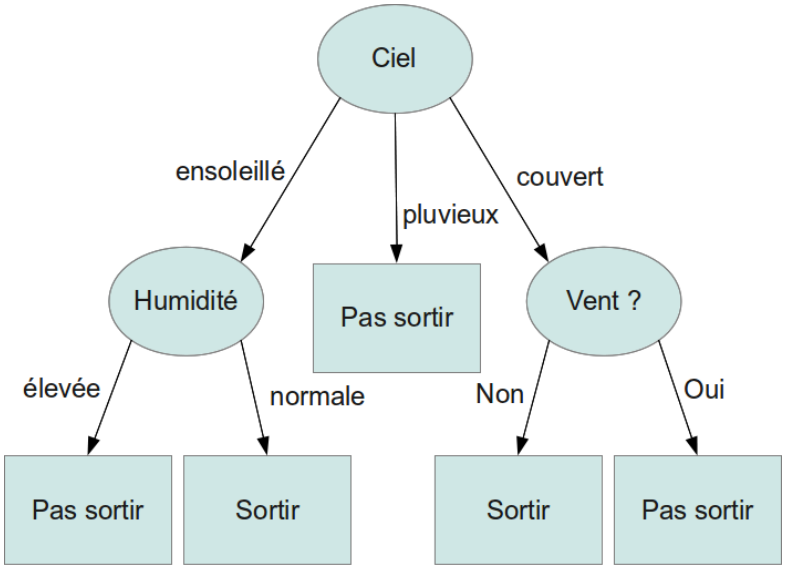


FIGURE 1.13 – Arbre de décision pour le problème de la météo

Il s'agit évidemment là d'une application très peu intéressante au vu de l'utilité de la réponse souhaitée mais il n'y a absolument aucune restriction quant aux données utilisées. Il existe des approches spécialisées du data mining dans les données météorologiques et qui permettent d'obtenir des modèles prévisionnels de la météo future avec un taux d'erreur tout à fait acceptable [21]. Le data mining est également utilisé dans bien d'autres domaines comme le marketing, la vente, la relation clientèle, la prévision de la charge électrique nécessaire à produire par une entreprise, le diagnostique de machines (savoir quand elles ont un dysfonctionnement), et beaucoup d'autres [20].

1.4.2 Reconnaissance de caractères manuscrits

Un système **OCR** (reconnaissance optique de caractères) est un système dont le but est de produire un fichier texte à partir d'une image scannée ou obtenue via une caméra numérique. Le but est donc de récupérer un document écrit à la main en sa version numérique afin de pouvoir l'utiliser directement sur l'ordinateur sans avoir à le recopier. La phase de reconnaissance des caractères utilise l'**apprentissage supervisé**, celui-ci étant utilisé de la manière décrite ci-dessous.

L'algorithme possède une bibliothèque de formes connues (donc des paires image/caractère représenté) grâce à laquelle il produit un modèle permettant de classifier n'importe quelle image selon plusieurs critères. Une fois ce modèle créé, chaque caractère obtenu depuis l'image est fourni au modèle et une classe lui est ainsi attribué, cette classe étant le caractère auquel l'image se rapprochait le plus [22]. Comme le montre la figure 1.14, il existe des exemples très simples comme il en existe des biens plus compliqués que même l'œil humain aurait quelques difficultés à déchiffrer. Cette dernière constatation explique pourquoi les exemples fournis sont très nombreux et qu'il y en a plusieurs par caractère.

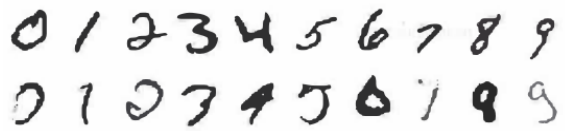


FIGURE 1.14 – Deux exemples de texte à reconnaître (tirés de AIMA [1])

1.4.3 Robotique

Programmer un robot mobile peut se révéler être une tâche très exigeante en temps. Il est même souvent difficile pour un programmeur de traduire sa connaissance quant à la manière de réussir une tâche en termes utiles au robot. Les senseurs et actuateurs des robots sont très différents de ceux des humains, une mauvaise conception peut causer un ratage complet du code de contrôle. L'idée de fournir une spécification haut-niveau de la tâche et utiliser l'apprentissage automatique pour "remplir les détails" est dès lors très intéressante.
En effet, il semble qu'il y a moins de temps perdu à écrire cette spécification haut-niveau ainsi que la politique de contrôle de l'apprentissage qu'à définir de manière très précise ce que le robot a à faire dans chaque situation [23]. Le robot apprend ainsi au travers de ses expériences. S'il tombe il comprend que ce qu'il a fait n'était pas une bonne idée et il se corrigera lors de la prochaine fois qu'il fera face à cette situation. Il s'agit là clairement d'une utilisation très intéressante de **l'apprentissage par renforcement** qui a été décrit en détails dans la section précédente.

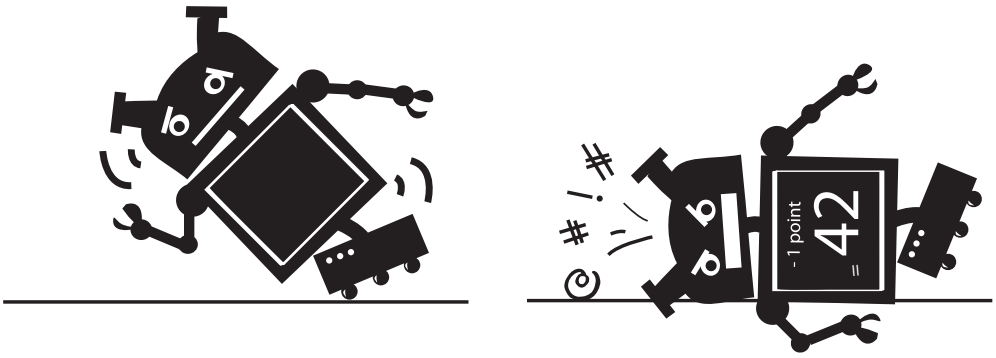


FIGURE 1.15 – Exemple de renforcement négatif pour un robot mobile

Chapitre 2

L'apprentissage automatique et la lecture de musique

Ce chapitre présente plusieurs applications existantes différentes qui font appel à l'apprentissage automatique afin de fournir une expérience unique à l'utilisateur désireux d'écouter de la musique en intervenant le moins possible. Il n'est pas toujours évident de parler de ces applications en détails car très peu d'entre elles sont *open-source*, c'est-à-dire que le code source n'est pas accessible et donc aucun indice de fonctionnement précis n'est disponible. Ce chapitre commence par une présentation de **XPod**, un système créé en université, pour ensuite s'intéresser à des logiciels plus connus du grand public comme **Apple Genius**, **Deezer**, **last.fm** et **Pandora** mais aussi des moins connus mais tout aussi intéressants comme **Jukefox**.

2.1 XPod

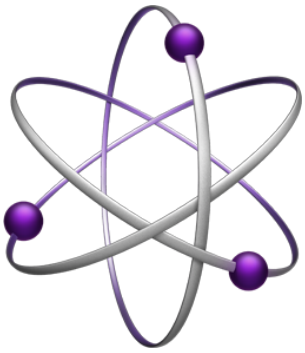
Il s'agit d'un système créé à l'université du Maryland [24] qui a pour but d'intégrer une conscience de l'activité humaine et de ses préférences musicales pour produire un système adaptatif qui lit une musique s'accordant bien au contexte. Ce projet introduit un lecteur de musique "intelligent" qui apprend des préférences et de l'activité de l'utilisateur et adapte en conséquence ses sélections de musique.

Le lecteur est associé à un dispositif permettant d'obtenir l'état physiologique de l'utilisateur. Ce dispositif permet, via les nombreuses variables dont il fait un monitoring, de déterminer le niveau d'activité de l'utilisateur de manière à décerner quelle musique il est approprié de lire. Parmi ces nombreuses variables figurent la température de la peau, la quantité de sueur sur la peau, l'accélération en deux dimensions (pour savoir s'il marche ou court) , ...

FIGURE 2.1 – Forme proposée pour le dispositif XPod

Au niveau du fonctionnement, **XPod** commence par une période d'entraînement avec l'utilisateur afin de cerner ses goûts musicaux (**apprentissage supervisé**). Après cette phase, le lecteur est capable d'utiliser ses algorithmes internes pour faire une sélection éduquée de la musique qui s'accorde le mieux avec l'émotion et l'activité actuelle de l'utilisateur. Avant de choisir une musique, l'algorithme interne prédit l'évaluation, l'appréciation de l'utilisateur sur cette musique dans l'état courant. Cette prédiction est utilisée pour pondérer la probabilité que la musique soit lue. Une musique dont l'évaluation estimée est basse peut être passée dans l'état courant mais ceci est fait pour que **XPod** explore tout l'espace des musiques et ne se cantonne pas à quelques unes.

2.2 Apple Genius



Pro-Apple ou pas, rares sont les personnes n'ayant jamais entendu parler de cette fonctionnalité "magique". Pour les personnes ne la connaissant pas, il s'agit d'une fonctionnalité du logiciel **iTunes** développé par **Apple** et qui est capable de vous présenter une liste de lecture de musiques **que vous possédez** et qui s'accordent bien ou encore une liste de musiques **que vous ne possédez pas** et que vous pourriez aimer. Sur ce point, ce que propose **Genius** est très différent de ce qui est proposé par le logiciel développé au cours de ce mémoire. En effet, **Genius** se base sur la

communauté entière de ses utilisateurs et non sur l'utilisateur unique.

FIGURE 2.2 – Apple Genius

Selon un article paru il y a deux ans [1], un ingénieur d'Apple, **Erik Goldman**, aurait vendu la mèche concernant le fonctionnement de cette fonctionnalité. Les fondements de cet article n'ayant pu être prouvés, il se peut que celui-ci soit erroné, il est donc de bon ton de prendre celui-ci avec des pincettes bien que tout ce qui y est dit semble très probable.

Comme il est précisé lors de l'activation de **Genius** (cf figure 2.2), la première phase du fonctionnement est de réunir des informations sur la façon dont l'utilisateur utilise iTunes. Parmi ces informations, il y a vraisemblablement et entre autres : quelle chanson l'utilisateur écoute, combien de fois une chanson est écoutée, quelles chansons sont passées lors de la lecture d'une liste de lecture, ... De manière basique, les informations collectées sont ensuite comparées à celles des autres utilisateurs de **Genius** via des algorithmes faisant appel à une forme d'analyse en composante principale (une forme d'**apprentissage non-supervisé**) en se basant sur un modèle d'espace vectoriel (chaque chanson est représentée comme un vecteur). L'idée de cette analyse en composante principale est assez simple dans les faits. En effet, il s'agit de réduire le problème à quelques variables. Ceci est rendu possible car souvent des informations récupérées n'influencent pas le résultat ou certaines informations sont redondantes (par exemple, les personnes qui sont tristes aiment les chansons tristes, inutiles de conserver les deux informations).

1. *"How iTunes Genius Really Works"* by Christopher Mims
http://www.technologyreview.com/blog/mimssbits/25267/

En résumé, et de manière simple, si vous écoutez souvent une musique A tout comme un certain nombre d'autres utilisateurs et qu'un bon nombre de ces utilisateurs écoutent également assez souvent une musique B ; **Genius** vous propose dans sa liste la musique B vu qu'elle pourrait rencontrer vos critères d'appréciation. Libre est le choix ensuite de lire la liste de lecture ou d'acheter certaines de ces musiques.

2.3 Jukefox

Jukefox [2], précédemment connu sous le nom de *museek*, fait partie d'un projet de recherche, nommé *musicexplorer*, en cours au sein du groupe d'informatique distribuée de **ETH Zurich**. Il s'agit d'une application pour smartphone Android permettant de lire de la musique, de l'organiser et de créer des listes de lectures intelligentes.

Une des deux fonctionnalités les plus intéressantes est que l'application *aide* l'utilisateur à trouver la musique particulière qu'il cherche. En effet, de nos jours, les lecteurs permettent la recherche d'une musique par des mots clés qui sont comparés aux metadata des fichiers de musique. Le problème est que, souvent, l'utilisateur ne se rappelle plus du titre de la musique ou de son auteur et est donc frustré de ne pas pouvoir retrouver cette musique qu'il aurait envie d'écouter.

Jukefox est équipé d'un mapping unique de similarité entre musiques qui reflètent les relations variées au sein de la bibliothèque de musiques de l'utilisateur. Ceci fournit donc un composant central pour comprendre les intentions de l'utilisateur. Basé sur cet outil très puissant, **Jukefox** expérimente de nouvelles façons d'accéder à une collection. Voici une liste des fonctionnalités intéressantes que ce logiciel fournit :

– **Smart shuffling**, il s'agit d'une fonctionnalité proche de celle recherchée par le lecteur développé dans ce mémoire. L'idée est de lire aléatoirement des musiques les unes après les autres tout en guidant la partie "aléatoire" vers des musiques qui sont susceptibles de plaire à l'utilisateur, de bien s'adapter à l'humeur de cet utilisateur. Pour ce faire, **Jukefox** considère que dès l'utilisateur passe une musique c'est qu'il ne veut pas écouter ce genre de musique pour le moment. De manière similaire, écouter une musique jusqu'au bout signifie que le genre de la musique plaît à l'utilisateur et conforte le lecteur dans le genre de musique qui est entrain d'être écouté.

2. www.jukefox.org

– **Play similar songs**, il s'agit de la fonctionnalité principale du lecteur développé dans ce mémoire. L'idée est de partir d'une chanson initiale (choisie aléatoirement ou choisie par l'utilisateur) et de lire consécutivement des musiques qui sont proches les unes des autres. **Jukefox** utilise pour ce faire son fameux mapping entre les différentes musiques qui est stocké localement et ne requiert donc aucune connexion internet (contrairement à Apple Genius).
– **Tag cloud**, il s'agit d'associer des tags sociaux aux musiques afin de faciliter les recherches. En effet, les genres de musique ne sont pas assez précis et performants pour les recherches car soit l'utilisateur n'est pas spécialement d'accord avec l'information renseignée ("ce n'est pas du rock ça c'est du pop-rock !") soit l'information est manquante ou encore plusieurs genres pourraient être associés à certaines musiques. **Jukefox** comprend tout à fait ces tags sociaux mais tag de lui-même les musiques (ces tags sont éditables) selon ce qui semble être le plus adapté.
– **Music map**, il s'agit réellement d'une carte de musiques. En fait les albums sont regroupés par région dans lesquelles tous les albums sont censés proposer du contenu similaire. Il est possible de (dé)zoomer afin de visualiser la bibliothèque utilisée de manière plus significative et plus intuitive qu'une simple liste alphabétique. Il est également possible de cliquer sur une région et de lancer la lecture d'une liste de lecture contenant quelques albums de cette région.

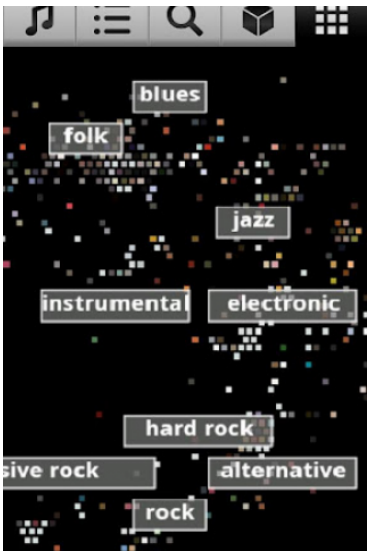


FIGURE 2.3 – Music map de Jukefox
(chaque point lumineux est un album de musiques)

En lisant la publication liée à la création de **Jukefox** [3], il apparaît que la notion utilisée est le **PLSA** (Probabilistic Latent Semantic Analysis) qui elle-même fait appel à l'algorithme **EM** (Expectation Maximisation) qui est un algorithme très connu dans le domaine de l'**apprentissage non-supervisé**. Cette publication explique avec plus ou moins de détails la manière de fonctionner utilisée par le logiciel pour trier et tagger les musiques.

2.4 Deezer et last.fm

Applications web permettant d'écouter de la musique gratuitement (enfin presque, l'écoute étant limitée tant qu'un compte payant n'a pas été activé) **deezer** [4] et **last.fm** [5] sont des "standards" en la matière et ont une notoriété assez importante. Il s'agit d'applications propriétaires et non open-source ce qui empêche des informations de filtrer concernant leur fonctionnement, d'où la pauvreté de cette section. Cependant, le fonctionnement de leur fonctionnalité "*radio*" est similaire voire identique et semble être également basé sur un système de tags sociaux à l'instar de **Jukefox** mais cette fois au travers d'une communauté, pas uniquement via l'utilisateur.

Pour l'utilisateur, le fonctionnement de la radio est simple, il suffit d'entrer le nom d'un artiste pour obtenir une suite de musiques *proches* de celles faites par l'artiste. L'algorithme derrière tout ça regroupe plusieurs musiques dont les tags sont proches de ceux de l'artiste sélectionné. Ce regroupement peut se faire sur le genre de la musique mais aussi sur des critères plus poussés comme le fait que le batteur du groupe A soit le chanteur du groupe B. Bien qu'aucune preuve n'a pu être trouvée, il est pratiquement certain que le fameux algorithme soit un algorithme d'**apprentissage** (supervisé et/ou non-supervisé voire semi-supervisé).

2.5 Pandora

De manière semblable aux webradios précédemment introduites, **Pandora** [6] permet, à partir de la sélection d'un artiste ou d'une musique en particulier, d'obtenir une succession de musiques s'y raccordant particulièrement bien. Il s'agit donc, comme la plupart des logiciels dans le domaine, d'un moyen de découvrir des nouvelles musiques susceptibles de plaire à l'utilisateur.

3. http://www.disco.ethz.ch/publications/mmfat11301-kuhn_221.pdf
4. www.deezer.com
5. www.lastfm.fr
6. www.pandora.com

Au niveau du fonctionnement de l'application du point de vue de l'utilisateur, ce dernier a, pour chaque titre qu'il écoute, plusieurs actions possibles :

- **Thumbs up**, "*j'aime cette musique*", signifier que jouer des musiques similaires à celle-ci est une bonne idée,
- **Thumbs down**, "*je n'aime vraiment pas cette musique*", signifier que jouer des musiques similaires à celle-ci est une très mauvaise idée et qu'il faudrait même éviter de jouer cette musique en particulier,
- **Zzzz**, signifier que cette musique n'est pas trop désirable pour l'instant, elle ne devrait plus être jouée pendant un moment (un mois),

Il peut également ne donner aucun feedback, dans ce cas **Pandora** ne prend aucune disposition particulière. Ces actions laissent penser que des techniques d'**apprentissage par renforcement** sont utilisées.

Le "Music Genome Project". Selon un article de *HowStuffWorks*[25], contrairement à bon nombre de ses concurrents (si pas tous), **Pandora** n'a pas de concept de genre, connexions entre utilisateurs ou d'évaluation de musiques. Le système utilise une approche radicalement différente : il analyse les structures musicales présentes dans les musiques que l'utilisateur aime et lit ensuite d'autres musiques qui ont des traits musicaux similaires. Ce système se base sur un "Music Genome" qui consiste en un ensemble de 400 attributs musicaux parmi lesquelles :

- la qualité de la mélodie,
- l'harmonie,
- le rythme,
- la composition,
- les paroles,
- ...

Ce génome est basée sur une analyse complexe des musiques existantes et ayant été crées ces 100 dernières années (il sera bientôt étendu aux musiques classiques) et cette analyse continue de jour en jour à chaque fois qu'une nouvelle musique fait son apparition. Bien qu'aucune information claire à ce sujet n'est mentionnée, il s'agit là vraisemblablement d'une application d'un algorithme d'**apprentissage supervisé**. En effet, les traits de musique sont entrés **manuellement** par un humain et ce, pour chaque musique contenue par la base de données. Ces traits enregistrés par le génome sont différents selon le genre de musique étudié. Ainsi une musique de rap sera bien plus étudiée au niveau de ses paroles tandis qu'une musique électronique dans son rythme ou sa musicalité.

Le petit plus de **Pandora**, c'est que lorsqu'il choisit une musique similaire, il explique à l'utilisateur les raisons qui l'ont mené à ce choix. Comme par exemple "la tonalité est semblable et il s'agit également d'un groupe dont le chanteur principal est un homme". Le petit moins cependant est qu'il n'est pas accessible en dehors du sol américain (à moins d'utiliser un serveur proxy ou autre méthode pour contourner l'interdiction). De plus, au niveau des interdictions et limitations,

il faut également compter l'impossibilité de jouer une musique spécifique et l'impossibilité de passer plus de 10 musiques par heure (pour ne pas pouvoir passer les chansons jusqu'à celle désirée). Il est, aussi, impossible de revenir en arrière dans la liste de lecture et le nombre de lectures d'une musique spécifique est limité pour une période de temps spécifique. Tout ceci enlève un peu de son sublime à l'expérience (bien entendu, moyennant une somme ronde et trébuchante, ces limitations peuvent tomber).

2.6 Comparatif

XPod est absent de ce comparatif car il n'a pas été possible de le tester.

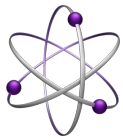
					
Apprentissage	Non-supervisé	Non-supervisé	?	?	Supervisé
Base	Communauté	Musiques	Musiques	Musiques	Musiques
Disponibilité					
Renforcement					
Open-source					
Documentation					
Gratuité					
Création listes					
Lecture aléatoire					
Découverte					

FIGURE 2.4 – Tableau comparatif des applications

La figure 2.4 contient un tableau comparatif des différentes applications présentées. Les deux premiers critères n'en sont pas, il s'agit des caractéristiques des applications. Pour les critères suivants :

– **Disponibilité** : les trois applications du type "web-radio" ne fonctionnent, comme leur nom l'indique, que lorsqu'un accès à Internet est disponible. **Apple Genius** fonctionne quant à lui, pour certains fonctionnalités, de manière totalement hors-ligne et pour d'autres, une connexion Internet

est nécessaire. **Jukefox** est la seule application fonctionnant totalement hors-ligne et qui est donc disponible en permanence.
- **Renforcement** : les trois "web-radios" sont les seules à permettre une forme de renforcement via un système de "J'aime/je n'aime pas".
- **Open-source** : la seule application open-source est **Jukefox**, ce qui est un énorme avantage pour elle.
- **Documentation** : ce critère découle du précédent. Il est possible de trouver des articles sur les idées utilisées par **Pandora** et **Apple Genius**. Cependant pour **last.fm** et **Deezer**, aucune information ne transparaît. La palme d'or revient une nouvelle fois à **Jukefox** dont les idées utilisées sont expliquées dans un article disponible sur le site officiel.
- **Gratuit** : à part **Jukefox**, toutes les applications ont au moins une petite partie payante. **Apple Genius** propose, entre autres, de nouvelles musiques mais elles doivent être achetées. Les trois "web-radios" quant à elles limitent leurs fonctionnalités ou le temps d'écoute dans leur version gratuite.
- **Création de listes de lecture** : toutes les applications le permettent, avec un petit plus pour **Jukefox** et **Apple Genius** qui permettent également de créer des listes intelligentes de manière automatique.
- **Lecture aléatoire** : ce critère est basé sur la fonctionnalité principale recherchée pour ce mémoire : il s'agit de partir d'une musique initiale et d'ensuite lire des musiques les unes après les autres de manière à ce qu'elles "aillent bien ensemble". Dans cette optique **Apple Genius** est totalement absent (même si la création d'une liste pourrait s'en rapprocher) et les "web-radios" s'en rapprochent mais ne permettent pas de considérer l'ensemble de la bibliothèque de musiques disponibles. **Jukefox** permet exactement cette fonctionnalité bien qu'il ne se base uniquement que sur les corrélations réelles entre les musiques (ou sur des tags sociaux) mais pas sur les actions de l'utilisateur.
- **Découverte** : **Jukefox** ne fonctionne qu'avec la bibliothèque de l'utilisateur et ne permet donc pas une découverte de nouvelles musiques. Les autres applications quant à elles permettent cette découverte et il s'agit d'une de leurs fonctionnalités principales.

En bref, l'application se rapprochant le plus des critères énoncés et de l'application qui est développée au cours de ce mémoire est **Jukefox**. Celle-ci est un modèle pour ce développement au niveau des fonctionnalités offertes. Il faut cependant remarquer que là où **Jukefox** se base entièrement sur ce qu'il a calculé via un apprentissage non-supervisé sur la bibliothèque de l'utilisateur, **SmartPlayer** utilise l'apprentissage par renforcement afin de faire évoluer ses données initiales et tendre vers les goûts réels de l'utilisateur. Il est à préciser qu'il existe bien d'autres applications qui n'ont pas été présentées comme **Grooveshark**, **Spotify** ou encore **Clementine**. Leur fonctionnement étant fort proche voire iden-

tique à une ou plusieurs applications présentées, il n'a pas été jugé pertinent de les développer davantage.

Chapitre 3

SmartPlayer

Ce chapitre présente le travail pratique exécuté au cours de ce mémoire. Il introduit l'application de lecture de musique intelligente **SmartPlayer** en expliquant son fonctionnement, le but recherché ainsi que sa conception et son implémentation dans le langage JAVA. Il présente ensuite différents tests et résultats obtenus lors de l'utilisation de l'application.

3.1 Présentation

Cette section a pour but de présenter l'application au travers d'un descriptif fonctionnel et d'explications en détails de chacune de ces fonctions. Elle commence par une introduction brève pour ensuite présenter en détails l'agent apprenant développé ainsi que chaque concept auquel ce dernier fait appel.

3.1.1 L'application

SmartPlayer est, comme son nom l'indique, un lecteur de musiques "intelligent" en ce sens où il est capable de comprendre les "désirs de l'utilisateur" au travers des actions que ce dernier effectue et des informations qu'il fournit. Pour y parvenir, le logiciel est majoritairement composé d'un agent d'intelligence artificielle apprenant au travers de l'apprentissage automatique à la fois non supervisé et par renforcement. Il présente également une interface graphique sommaire permettant une expérience plus intéressante que via une application orientée "console" mais ceci ne consiste en rien son but. Il est aussi doté d'un système de log entièrement basé sur l'utilisation de la *programmation orientée aspects* dont une explication détaillée est donnée dans la section 3.3.3.

Point de vue fonctionnel, l'application doit donc être capable de lire des musiques (limité aux fichiers au format MP3 dans le cadre de ce mémoire), tant spécifiées par l'utilisateur qu'aléatoirement choisies parmi les musiques existantes dans sa bibliothèque. Elle doit en outre construire au fur et à mesure une liste de

lecture dans laquelle l'utilisateur peut se déplacer en avant et en arrière. Le choix aléatoire de musiques doit être dirigé pour correspondre aux envies de l'utilisateur afin que l'expérience vécue par celui-ci soit la plus optimale possible. Finalement, elle doit permettre de faire les actions de base d'un lecteur de musiques : play, pause, stop, musique suivante, musique précédente.

3.1.2 L'agent intelligent

Le principal but de l'agent est, bien entendu, de contenter le plus possible l'utilisateur. Ce contentement s'obtient principalement par la sélection, à partir d'une musique donnée, de la musique qu'il faudrait jouer ensuite. Il s'agit donc de déterminer quelle musique serait la plus pertinente (pour l'utilisateur) de suivre celle en cours de lecture. Pour atteindre ce but, l'agent intelligent est composé de deux parties :

1. **l'apprentissage initial**, celui-ci consiste en un clustering sur les musiques afin de regrouper celles qui semblent **similaires** de prime abord,
2. **l'apprentissage par renforcement**, celui-ci constitue réellement l'intelligence et le point d'intérêt de l'application, c'est cette partie qui se montre capable d'interpréter les actions de l'utilisateur pour ensuite faire des choix en conséquences.

Bases d'informations de l'agent

Avant de développer les algorithmes d'apprentissage utilisés par l'agent, un petit coup d'œil aux différentes informations intéressantes stockées par l'agent. Les informations proviendront essentiellement de deux sources :

– **Une base de données** M_i contenant les musiques de la bibliothèque de l'utilisateur, elle contiendra pour chaque musique : un identifiant (le i utilisé pour indicer M) ainsi que les informations importantes comme l'auteur, le titre ou le genre ainsi qu'un attribut nommé "*appréciation*". Ce dernier représente l'appréciation globale de l'utilisateur pour la musique concernée, notion abordée avec plus de précisions dans la section 3.1.4.

id	author	title	genre	duration	release_date
1	Arid	If you go	Rock	4 :10	2012-01-01
2	Stromae	Alors on danse	Pop	3 :27	2011-14-09
⋮	⋮	⋮	⋮	⋮	⋮
n	Laura Pausini	La solitudine	Romantic	3 :55	2012-03-04

path	copyright	comment	appreciation
musics/Arid/If you go.mp3	None	Belgian artist !	1.00
musics/Stromae/Alors on danse.mp3	None	None	0.75
⋮	⋮	⋮	⋮
musics/Romantic/Laura Pausini - La solitudine.mp3	None	Downloaded with iTunes	0.5

FIGURE 3.1 – Exemple de base de données M_i

– **Une matrice triangulaire supérieure** S_{ij} *(sans la diagonale)* contenant les *similarités* entre chaque couple possible de musiques (en évitant les redondances). Cette notion de similarité consiste en un nombre représentant à quel point deux chansons sont semblables, elle est abordée plus en détails dans la section 3.1.3.

$$S_{ij} = \begin{pmatrix} 0 & similarity(1,2) & similarity(1,3) & \cdots & similarity(1,n-1) & similarity(1,n) \\ 0 & 0 & similarity(2,3) & \cdots & similarity(2,n-1) & similarity(2,n) \\ 0 & 0 & 0 & \cdots & similarity(3,n-1) & similarity(3,n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & similarity(n-2,n-1) & similarity(n-2,n) \\ 0 & 0 & 0 & \cdots & 0 & similarity(n-1,n) \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

FIGURE 3.2 – Structure de la matrice S_{ij}

L'ensemble des hypothèses est donc ici l'ensemble des matrices triangulaires de nombres réels pour les similarités et l'ensemble des matrices à une seule ligne de nombre réels pour les appréciations. Ces ensembles ont été choisis car ils permettent de bien représenter le problème et qu'ils sont simples à comprendre et à utiliser.

Clustering

Il s'agit de l'étape initiale de l'apprentissage de l'agent. A chaque nouvelle musique ajoutée, le clustering est relancé afin de déterminer une estimation de la similarité théorique de cette musique avec toutes les autres musiques déjà contenues dans la base de données. L'algorithme de clustering utilisé est un algorithme de clustering hiérarchique agglomératif, un peu modifié, utilisant comme mesure de similarité entre cluster le critère **group-average**.

Cette adaptation est très simple, elle consiste à éviter d'obtenir des valeurs trop spécifiques pour chaque couple de musiques, c'est-à-dire que, pour définir la similarité entre A et B, l'algorithme va tenter de tirer profit de la similarité de A avec d'autres musiques. La raison à cela est illustrée dans l'exemple ci-dessous.

Exemple 3.1.1. *trois musiques A, B et C telles que :*

- $similarity(A,B) = 0.9$
- $similarity(A,C) = 0.3$
- $similarity(B,C) = 0.9$

Si aucune adaptation n'est appliquée, ces similarités seront donc prises telles quelles et cette situation semble très peu réaliste. En effet, comment cela se peut-il que A soit très proche de B et que B soit très proche de C alors que A et C

sont très différentes ? L'application du clustering permet ainsi d'uniformiser ces similarités.

Pour ce faire, l'algorithme groupe toutes les musiques par clusters de manière hiérarchique jusqu'à obtenir un seul cluster (au départ chaque musique définit son propre cluster). A chaque fusion de deux clusters, la similarité de tous les couples tels que le premier élément est pris dans le premier cluster et le second dans le second cluster est initialisée à la valeur de similarité *group-average* entre ces deux clusters. De plus, afin de ne pas interférer avec l'apprentissage par renforcement, l'algorithme ne modifie la valeur de similarité que si celle-ci n'était pas encore définie. L'algorithme 1 reproduit ce comportement.

Algorithme 1 Clustering agglomératif adapté

Entrée :
- M, la liste des n musiques,
- S, la matrice triangulaire des similarités.

Sortie : /

$clust \leftarrow$ liste de n clusters vides
Pour chaque musique m_i de $musics$ **faire**
  Ajouter m_i à $clust_i$
fin Pour
Tant que $clust$ contient plus d'un cluster **faire**
  $sim_{max} \leftarrow -1$
  / Recherche des deux clusters les plus similaires */*
  Pour chaque cluster c_i de $clust$ **faire**
    Pour chaque cluster c_j de $clust$ avec $j > i$ **faire**
      $sim_{current} \leftarrow group - average(c_i, c_j)$
      Si $sim_{current} > sim_{max}$ **alors**
        $sim_{max} \leftarrow sim_{current}$
        $first\_cluster\_to\_merge \leftarrow c_i$
        $second\_cluster\_to\_merge \leftarrow c_j$
      fin Si
    fin Pour
  fin Pour
  / La similarité entre chaque couple possible = similarité group-average entre les deux clusters */*
  Pour chaque musique m_i de $first\_cluster\_to\_merge$ **faire**
    Pour chaque musique m_j de $second\_cluster\_to\_merge$ **faire**
      Si S_{ij} n'a pas déjà été initialisé **alors**
        $S_{ij} \leftarrow sim_{max}$
      fin Si
    fin Pour
  fin Pour
  merge($first\_cluster\_to\_merge$*,*$second\_cluster\_to\_merge$*)*
fin Tant que

Apprentissage par renforcement

Comme expliqué plus haut, il s'agit ici de la partie la plus importante de l'agent. Cette partie va permettre à ce dernier d'apprendre et de se rapprocher des goûts de l'utilisateur au fur et à mesure que celui-ci utilise l'application. Le rôle de cet apprentissage est de faire évoluer les coefficients de similarité et d'appréciation au cours du temps. Pour définir un agent apprenant par renforcement, il convient de définir ses différentes parties (comme explicité dans la section 1.3.4) :

- **La politique** *ϵ-greedy* a été privilégiée car plus simple à concevoir et à paramétrer.
- **Les senseurs**, ils sont au nombre de deux dont un n'est pas toujours utilisable. En effet, ce dernier permet d'obtenir des informations de la part de l'utilisateur via le *"Mood Sensing"* dont le fonctionnement est expliqué dans le paragraphe suivant. L'autre senseur, plus conventionnel, permet quant à lui d'obtenir l'action effectuée par l'utilisateur. Le senseur perçoit donc des couples (action,humeur).
- **Les actuateurs**, en outre l'actuateur permettant de modifier la musique devant être lue, d'autres actuateurs plus internes permettent de mettre à jour les coefficients. Ces actuateurs utilisent un mapping entre un couple (action,humeur) perçu par les senseurs et une action d'augmentation ou de diminution de la similarité entre deux musiques ou de l'appréciation d'une musique.

Actions\ Humeur	Content	Indifférent	Mécontent	Inconnu
Next (milieu d'une musique)	Augmentation de la similarité		Diminution de la similarité	
	Diminution de l'appréciation de la chanson passée			
Next (fin d'une musique)	Augmentation de la similarité		Diminution de la similarité	
	Augmentation de l'appréciation de la chanson passée			
Previous	Augmentation l'appréciation de la chanson précédente			

FIGURE 3.3 – Mapping (Action,Humeur) $\rightarrow$ Diminution ou augmentation

Comme décrit par le tableau ci-dessus, que l'utilisateur soit content ou pas, lorsqu'il passe une musique l'agent considère qu'il ne l'aime pas. Il s'agit là d'une hypothèse assez plausible vu qu'il s'agit de la raison principale pour laquelle un utilisateur passe une musique avant que celle-ci ne soit terminée.

- **La fonction de récompense**. Cette fonction est définie pour obtenir la similarité entre la musique précédente et la musique lue à présent ainsi que l'appréciation de cette dernière en guise de récompense. Elle donne également un point supplémentaire lorsque l'utilisateur s'est montré content des actions de l'agent et un point de pénalité quand celui-ci s'est montré mécontent.

De manière résumée :

$$f(\text{“}\textit{happy}\text{”}) = 1 \qquad f(\text{“}\textit{unhappy}\text{”}) = -1$$
$$f(\text{“}\textit{unknown}\text{”}) = 0 \qquad f(\text{“}\textit{indifferent}\text{”}) = 0$$
$$f(a,b) = sim(a,b) + appreciation(b) \quad \text{où } a \text{ et } b \text{ sont des musiques différentes.}$$

– **La fonction de valeur** consiste en une simple prévision de la récompense moyenne obtenue en effectuant des choix gloutons lors d'un nombre spécifié d'itérations appelé *profondeur*. Ainsi si la profondeur est 5, la fonction de valeur calcule la récompense moyenne obtenue si les 5 prochaines actions sont optimales et ce, en partant de chaque action possible. La manière de procéder de cette fonction est disponible dans l'annexe B.1.

L'algorithme 2 retrace le comportement adopté par l'agent afin de déterminer la prochaine musique à lire.

Algorithme 2 ϵ-greedy_GetNextSong

Entrée :
- M, la liste des n musiques ordonnée par appréciation décroissante,
- S, la matrice triangulaire des similarités,
- $previous\_song$, la musique qui était lue avant la demande de changement,
- ϵ, la probabilité d'agir de manière non-gloutonne

Sortie : la musique à jouer à présent.

$val_{max} \leftarrow$ -42
$j \leftarrow previous\_song_{id}$
Si la politique doit agir de manière gloutonne ($1 - \epsilon$ de probabilité) **alors**
  Pour toutes les musiques $m_i \neq m_j$ de M **faire**
    $val_{current} \leftarrow f_{\text{VALUE}}(i)$
    Si $val_{current} > val_{max}$ **alors**
      $val_{max} \leftarrow val_{current}$
      $music \leftarrow m_i$
    Sinon Si $sval_{current} = val_{max}$ et qu'un booléen aléatoire est vrai **alors**
      $val_{max} \leftarrow val_{current}$
      $music \leftarrow m_i$
    fin Si
  fin Pour
Sinon
  $music \leftarrow$ une musique qui n'a pas encore été lue au hasard
fin Si
Retourner $music$

Mood Sensing

Le **Mood Sensing** consiste à demander à l'utilisateur si le choix que l'agent a fait lui convient ou ne rentre pas du tout dans ses critères de préférences. Ainsi, lorsqu'une musique se termine et qu'une autre commence, un pop-up s'affiche et l'utilisateur a le choix entre trois "niveaux de joie" : **content**, **indifférent** et **pas content**. Ces niveaux de joie ou *"humeur"* permettent à l'agent de déterminer si son choix était bon ou mauvais et ainsi mettre à jour les coefficients d'appréciation et de similarité en conséquence. Cette fonctionnalité est totalement optionnelle (car elle peut être invasive pour l'utilisateur). L'utilisateur a le choix d'ignorer le pop-up (il se ferme automatiquement après 5 secondes) lorsqu'il n'a pas envie de répondre, conservant ainsi un moyen d'interagir avec l'agent lorsqu'une chanson aura particulièrement été bien choisie ou à l'inverse très mal choisie. Il peut également désactiver totalement la fonctionnalité (qui peut être réactivée via un menu).

Résumé / Schéma

L'algorithme 3 et la figure 3.4 résument le fonctionnement de l'agent lorsque l'application est en cours d'exécution. Il ne faut pas perdre de vue que lorsqu'une nouvelle musique est ajoutée, le clustering est lancé afin de calculer les similarités entre la musique ajoutée et toutes les autres musiques déjà intégrées.

Algorithme 3 SmartPlayerAgent

Entrée :
- M, la liste des n musiques ordonnées par appréciation décroissante,
- S, la matrice triangulaire des similarités,

Sortie : /

Tant que l'application tourne **faire**
  Tant que aucune action n'est ressentie **faire**
    attendre de manière non-active.
  fin Tant que
  action, humeur ← *sensors*.getSensedValues()
  todos ← policy.getActionsToDo(*action*,*humeur*)
  / todos contient des actions comme passer la prochaine musique choisie, augmenter une similarité, ... */*
  Pour toutes les actions *a* contenues dans *todos* **faire**
    actuators.doAction(*a*)
  fin Pour
fin Tant que

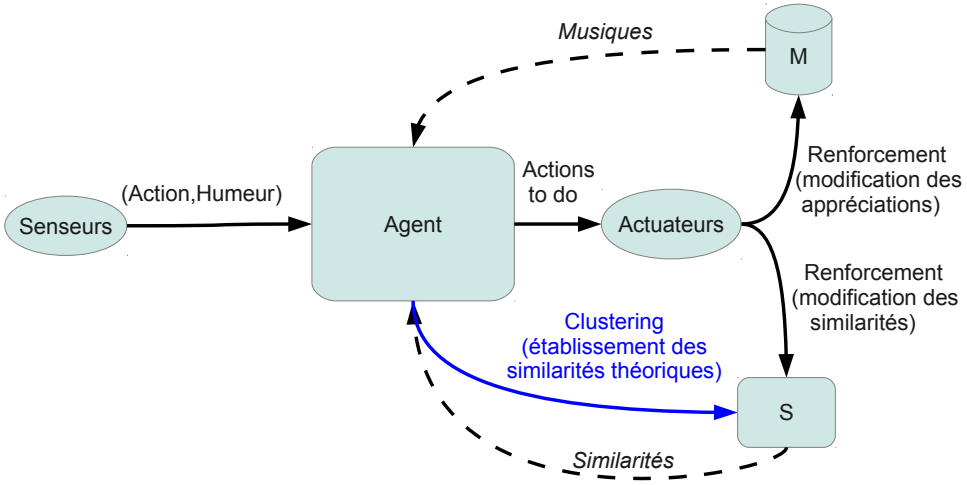


FIGURE 3.4 – Résumé du fonctionnement de l'agent

3.1.3 Similarité

La similarité initiale telle que définie par le clustering (Section 3.1.2) est une notion importante qui peut déterminer totalement le comportement de l'algorithme de clustering. Cette notion se base sur les tags ID3 des fichiers MP3 afin d'évaluer la ressemblance théorique entre deux musiques. Deux solutions sont alors envisageables : définir une similarité de manière empirique pour chaque couple de musiques possible (ou de manière un peu plus générale en ne se basant que sur le genre et peut-être sur l'artiste) ou trouver une mesure permettant de définir une similarité entre des chaînes de caractères. C'est la seconde approche qui s'est imposée car bien plus générale et demandant moins de travail bien qu'elle soit sans doute moins précise. Il existe différentes distances définies pour calculer la similarité de chaînes de caractères mais une seule semble vraiment intéressante dans le cas présent. En effet, la distance dont l'algorithme a besoin est une manière de découvrir que des musiques ont des genres, artistes et/ou titres fort proches (par exemple "Hard Rock" doit être proche de "Rock" mais pas totalement semblable). La distance qui est utilisée dans ce mémoire est donc la *distance de Jaro* [1].

1. http://fr.wikipedia.org/wiki/Distance_de_Jaro-Winkler#Distance_de_Jaro

Distance de Jaro. Cette distance est une mesure de similarité entre deux chaînes de caractères, plus elle est élevée plus les chaînes sont similaires. Cette distance est normalisée afin de ne couvrir que l'intervalle $[0, 1]$. Elle repose sur le calcul de plusieurs coefficients afin de déterminer sa valeur donnée par :

$$d_j(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$$

où :

- $|s_i|$ désigne la longueur de s_i,
- m désigne le nombre de caractères présents dans chacune des deux chaînes,
- t désigne le nombre de *transpositions*.

Deux caractères sont considérés comme **correspondants** si leur éloignement ne dépasse pas un certain seuil dépendant des tailles des chaînes de caractères :

$$threshold = \left\lfloor \frac{\max(|s_1|, |s_2|}{2} \right\rfloor - 1$$

Le nombre de transpositions est obtenu en comparant la séquence des caractères correspondants de s_1 et celle de s_2. En comparant position par position, le nombre de fois où les deux caractères sont différents divisé par 2 donne le nombre de transpositions.

Exemple 3.1.2. *Soit les chaînes de caractères suivantes :*

- s_1 =*Metal*
- s_2 =*Black metal*
- s_3 =*Techno*

Calcul de la distance de jaro entre s_1 ***et*** s_2

- $threshold = \lfloor 11/2 \rfloor - 1 = 4$
- $m = 5$
- *la séquence de caractères correspondants de* s_1*,* s_2 *est : [A,L]*
- *la séquence de caractères correspondants de* s_2*,* s_1 *est : [L,A]*
- $t = \frac{2}{2} = 1$
- $d_j(s_1, s_2) = \frac{1}{3}\left(\frac{5}{5} + \frac{5}{11} + \frac{5-1}{5}\right) = 0.751515$

Calcul de la distance de jaro entre s_1 ***et*** s_3

- $threshold = \lfloor 11/2 \rfloor - 1 = 2$
- $m = 2$
- *la séquence de caractères correspondants de* s_1*,* s_2 *est : [E]*
- *la séquence de caractères correspondants de* s_2*,* s_1 *est : [E]*
- $t = \frac{0}{2} = 0$
- $d_j(s_1, s_2) = \frac{1}{3}\left(\frac{2}{5} + \frac{2}{6} + \frac{2-0}{2}\right) = 0.5777777$

Similarité entre deux musiques. Maintenant que la distance de Jaro a été définie, la distance/similarité entre deux musiques peut être définie. Cette distance est une somme pondérée de similarité/distance entre les différents tags des deux musiques.

Définition 3.1. *Soit deux musiques A et B, alors la similarité entre A et B est donné par :*

$$sim(A, B) = \sum_{t \in tags} w_t d_j(A_t, B_t)$$

et dans ce mémoire, les poids non-nuls donnent la formulation finale suivante :

$$\begin{aligned} sim(A, B) &= 0.5 \times d_j(A_{genre}, B_{genre}) \\ &+ 0.35 \times d_j(A_{author}, B_{author}) \\ &+ 0.15 \times d_j(A_{title}, B_{title}) \end{aligned}$$

(Ces coefficients ont été définis sur base d'expérimentations)

3.1.4 Appréciation

L'appréciation est un nombre réel compris entre 0 et 1 associé à une musique M. Cette appréciation détermine l'appréciation globale de cette musique par l'utilisateur. Plus celui-ci l'apprécie et donc l'écoute en entier ou la lit expressément plus cette appréciation est proche de 1. A l'inverse moins il l'apprécie et donc plus il la passe lors de sa lecture plus cette appréciation est proche de 0.

Variation. Afin de faire varier cette appréciation, il a fallu déterminer deux fonctions $f : [0, 1] \to [0, 1] : x \to y = f(x)$:

- la première est telle que $\forall x \in [0, 1[, x < f(x)$ et $x = f(x)$ ssi $x = 1$ et est utilisée pour augmenter l'appréciation,
- la seconde est telle que $\forall x \in]0, 1], x > f(x)$ et $x = f(x)$ ssi $x = 0$ et est utilisée pour diminuer l'appréciation.

Les fonctions choisies l'ont été suite à des expérimentations. Ces expérimentations ont permis de montrer que lorsque l'appréciation est aux alentours de 0.5 elle a tendance à rester un peu stable et s'il y a plusieurs augmentations/diminutions successives, l'impact de ces variations est de plus en plus marqué. Une fonction sigmoïde aurait pu être utilisée pour représenter ce comportement mais cela pose problème car les conditions $\forall x \in]0, 1], x > f(x)$ ou $\forall x \in [0, 1[, x < f(x)$ ne sont pas respectées. En conséquence, la détermination de ces deux fonctions s'est faite via l'interpolation polynomiale sur certains points clés comme $(1, 1)$ pour la fonction d'augmentation et $(0, 0)$ pour la fonction de diminution.

$$incr : [0, 1] \to [0, 1] : x \to incr(x) = \begin{cases} -0.8x^2 + 2x - 0.2 & \text{si } x \geq 0.5 \\ 1.04x^2 + 0.58x + 0.05 & \text{sinon} \end{cases}$$

$$decr : [0, 1] \to [0, 1] : x \to decr(x) = \begin{cases} 0.8x^2 - 0.2x + 0.35 & \text{si } x \geq 0.5 \\ -0.32x^2 + 1.04x & \text{sinon} \end{cases}$$

Ces fonctions sont dessinées dans les figures 3.5 et 3.6.

Remarque 3.1.1. *Ces fonctions sont également utilisées pour les variations de similarités.*

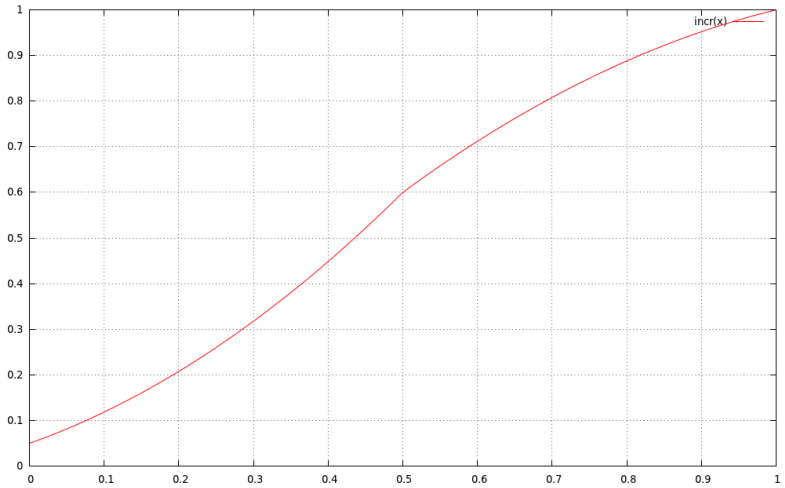


FIGURE 3.5 – Fonction d'augmentation de l'appréciation

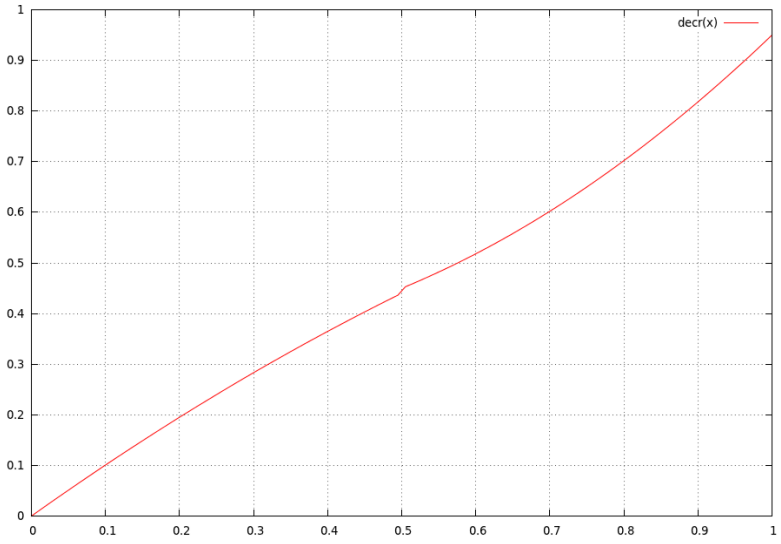


FIGURE 3.6 – Fonction de diminution de l'appréciation

3.2 Conception

Cette section trace la conception du logiciel en passant en revue ses différentes parties et les relations existant entre les classes JAVA. Elle commence par s'intéresser aux structures utilisées pour ensuite se concentrer sur les classes déterminant le lecteur. Par la suite, elle étudie les classes composant l'intelligence artificielle pour finir par les classes et aspects (voir section 3.3.3 pour une présentation des aspects) pour le système de log et les classes de l'interface graphique.

3.2.1 Structures

Il s'agit d'une partie primordiale car elle conditionne les performances de toute l'application qui manipule ces structures tout au long de son exécution. L'élément central est la classe **MP3File** qui, comme son nom l'indique, permet de représenter des fichiers MP3. Lors de la création d'une instance de cette classe, une exception peut être générée si le fichier utilisé n'existe pas ou si le fichier n'est pas au bon format. Pour clarifier ces différentes exceptions, une classe d'exception a été créé, il s'agit de **MP3Exception** qui peut être de deux types : FILE_NOT_FOUND et WRONG_FORMAT (symbolisé par l'enum **ExceptionType**).

La classe **MP3File** contient toutes les informations récupérables d'un fichier MP3 (via tags IDv3) comme l'auteur de la musique, le titre, la durée, etc... La durée est représentée par la classe **Duration** qui permet une gestion simple (et simplifiée par rapport à l'objet classique) de la durée d'une musique. Lorsque l'on regroupe plusieurs fichiers MP3 ensemble, et donc une liste de **MP3File**, on obtient une **MP3List** qui est une extension de la classe ArrayList de JAVA. De plus, une **PlayList** qui définit une liste de musiques à lire dans l'ordre est définie par un nom et une **MP3List**.

Pour construire des instances de ces structures de manière rapide, la classe **MP3FileFinder** a été implémentée. Celle-ci permet, depuis un répertoire donné, de rechercher et dresser une liste des fichiers MP3 trouvés. Il crée donc des objets **MP3File** correspondant aux musiques trouvées et les ajoute progressivement dans un objet **MP3List**.

Pour le clustering, il a fallu intégrer la notion de clusters de **MP3File**, ce qui est fait par la classe **MP3Cluster**. Celle-ci est également une extension d'ArrayList de JAVA mais propose des méthodes différentes comme le fait de fusionner avec un autre cluster. Le clustering fait aussi appel à la classe **Couple** permettant de représenter un couple et utilisant les "generics".

Finalement, toutes ces classes implémentent l'interface **Dumpable** qui permet de spécifier ce qu'il faut afficher lorsque la classe est loggée (plus d'informations dans la section 3.3.3). La figure 3.7 illustre les relations entre les classes liées à la classe **MP3File**.

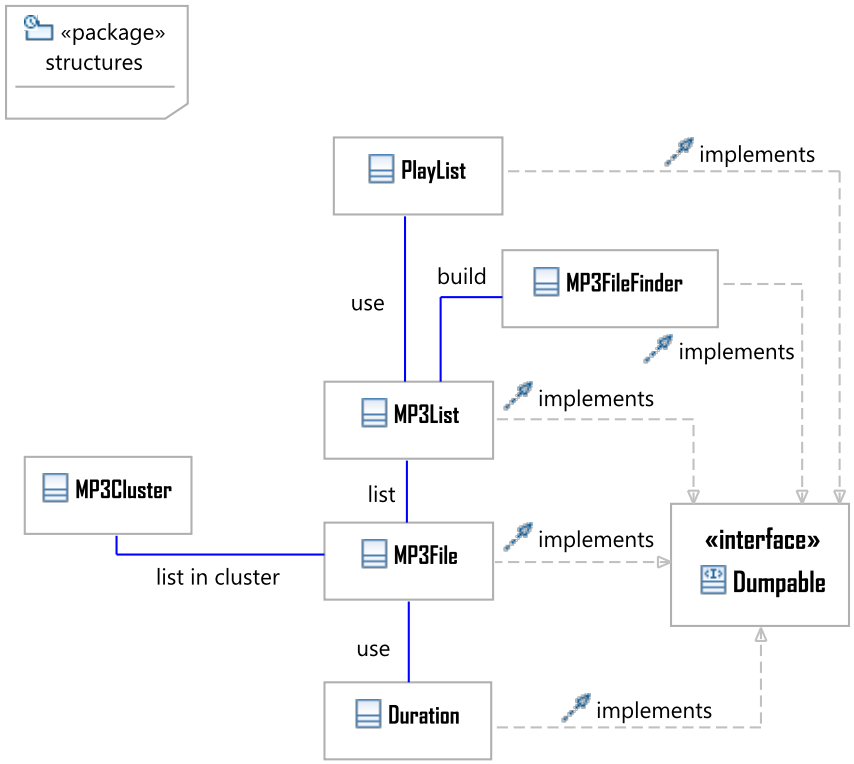


FIGURE 3.7 – Diagramme de classe des structures

3.2.2 Lecteur

Le lecteur utilisé par l'agent intelligent est défini par la classe **SmartPlayer**. Il est cependant possible d'implémenter son propre lecteur pour autant que celui-ci étende la classe abstraite **Player**. Cette dernière classe génère des événements **PlaybackEvent** lorsque les actions suivantes sont effectuées :

- **play**, le lecteur doit lire la musique courante,
- **pause**, le lecteur doit interrompre la lecture,
- **stop**, le lecteur doit stopper son fonctionnement,
- **prev**, le lecteur doit lire la précédente musique,
- **next**, le lecteur doit lire la musique suivante (il existe deux événements next, distingués grâce au booléen "complete" spécifiant si la musique passée a été lue jusqu'au bout).

Ces événements générés permettent à une classe implémentant l'interface **PlayerActionListener** de se tenir informer des actions effectuées par le lecteur. C'est typiquement le cas de la classe **SmartSensor** qui a besoin de savoir lorsque le lecteur change de musique.

Ces actions peuvent altérer l'état du lecteur. Ce lecteur peut effectivement exister en plusieurs états : *paused*, *playing* et *stopped* ; ceux-ci ayant un nom assez parlant. Les transitions entre ces différents états sont illustrés dans la figure 3.8.

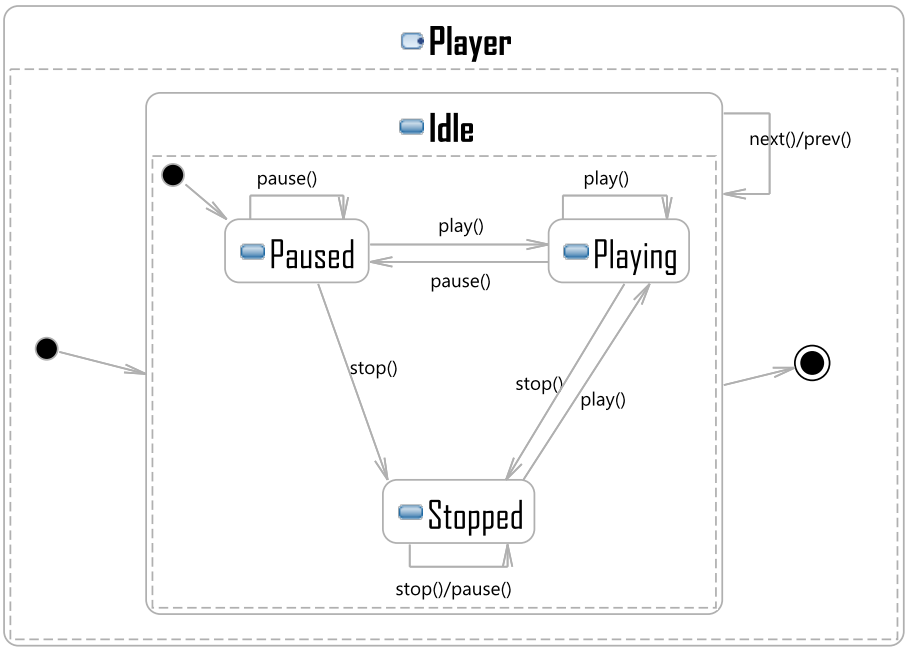


FIGURE 3.8 – Diagramme d'états du lecteur

Pour lire une musique, la classe **SmartPlayer** fait appel à une classe permettant de contrôler la lecture d'une musique : **SmartController**. Une autre classe pourrait être définie pour effectuer ce travail pour autant que la classe abstraite **PlayerController** est étendue. Cette dernière contrôle une classe nommée **SongPlayer** qui permet la lecture d'une seule musique, ainsi, lorsqu'une musique est passée, un nouveau **SongPlayer** est créé pour lire la nouvelle musique. Cette classe hérite d'une classe définie dans le module *JLayer* (plus d'informations dans la section 3.3.1) et permet d'y ajouter la possibilité d'interrompre et reprendre la lecture d'un fichier MP3. Cette classe est également écoutée par la classe **PlayerPlaybackListener** permettant d'obtenir le nombre de parties du fichier qui ont été lues si cela se révèle être nécessaire.

Concernant la lecture proprement dite, la classe **PlayerController** lance une tâche de lecture (classe **PlayTask**) permettant de lire le fichier MP3 dans un thread séparé. Cette lecture consiste ensuite à lire une infime partie du fichier MP3 (à peine audible) tant que le lecteur est dans l'état "playing". Dès que celui-ci sort de cet état pour l'état "paused", la lecture est interrompue via une synchronisation par sémaphore. Lorsque le lecteur retourne dans l'état "playing" le sémaphore est

utilisé pour relancer la lecture. Dans le cas de l'état "stop", la tâche libère les ressources utilisées et se met en attente de la future musique à faire lire par le lecteur.

Les relations entre ces différentes classes sont résumées sur la figure 3.9.

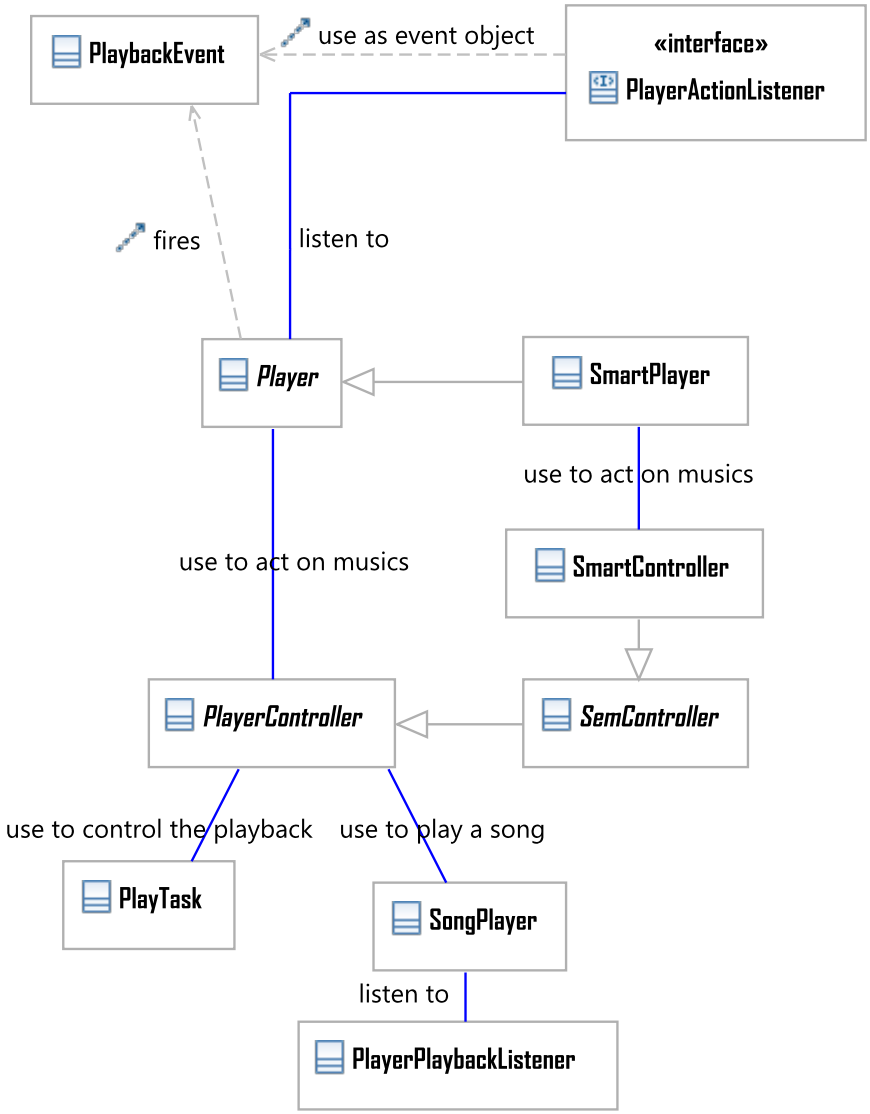


FIGURE 3.9 – Diagramme de classes du lecteur

3.2.3 Intelligence Artificielle

L'agent d'intelligence artificielle est représenté par la classe **SmartAgent**. Cette dernière interagit avec la classe **SmartPlayer** et utilise la classe abstraite **MP3StorageModel** afin d'interagir avec la collection des fichiers MP3. Une implémentation possible via l'utilisation de bases de données est fournie par la classe **DatabaseManager**.
Le **SmartAgent** est principalement un agent d'apprentissage par renforcement et possède donc la structure d'un tel agent (décrite plus tôt dans la section 1.3.4). Chacune de ses différentes parties est donc représentée par une classe comme décrit ci-dessous :

- La fonction de valeur représentée par la classe abstraite **ValueFunction** dont une implémentation complète est proposée dans **SmartValueFunction**.
- La fonction de récompense représentée par la classe abstraite **RewardFunction** dont une implémentation complète est proposée dans **SmartRewardFunction**.
- La politique représentée par la classe abstraite **Policy** dont une implémentation complète est proposée par **SmartPolicy**, celle-ci utilisant la politique ϵ-greedy. Elle effectue un mapping entre des couples de valeurs définis par les énumérations **Mood** et **Action** et un objet **Change** (dont le type est défini par l'énumération **ChangeType**) permettant de spécifier à l'actuateur le changement qui doit être opéré. La politique effectue également le clustering en utilisant les classes **MP3Cluster** et **SimilarityCalculator** qui permettent respectivement de grouper des musiques en clusters et d'obtenir la similarité entre deux musiques.
- Les actuateurs et les senseurs représentés respectivement par les classes **SmartActuator** et **SmartSensor**. Le premier utilise deux implémentations de l'interface **Function** afin de déterminer les incrémentations et décrémentations des valeurs de similarité et d'appréciation. Le second utilise les énumérations **Mood** et **Action** afin de définir les couples de valeurs ressenties. **SmartSensor** implémente également l'interface **PlayerActionListener** comme expliqué dans la section 3.2.2.

L'agent est également capable de détecter l'ajout d'une nouvelle musique et d'exécuter en conséquence le clustering afin d'estimer les valeurs de similarités entre cette nouvelle musique et les autres préalablement stockées. Les relations entre ces différentes classes sont illustrées dans la figure 3.10.

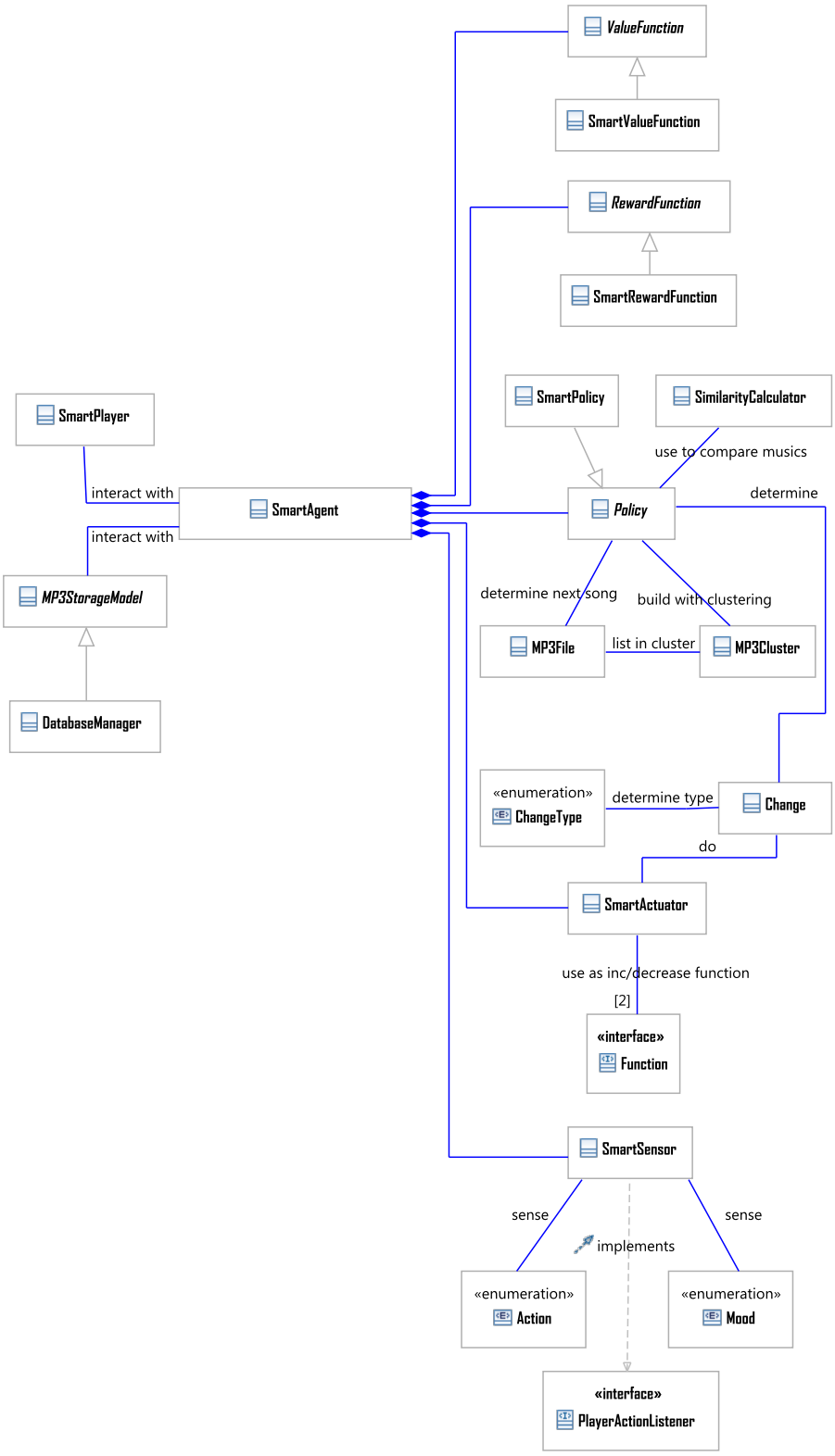


FIGURE 3.10 – Diagramme de classes de l'intelligence artificielle

3.3 Implémentation

Cette section trace les différentes étapes et parties importantes de l'implémentation de **SmartPlayer**. Elle s'intéresse ainsi aux librairies utilisées pour la lecture de fichiers MP3, au système de log utilisant la programmation orientée "aspects", le système de configuration ainsi que l'interface graphique.

3.3.1 Lire des fichiers MP3 - JLayer

JLayer est une librairie JAVA permettant de lire simplement un fichier MP3 donné. La dernière version date de 2008 et est disponible sur le site de leurs développeurs, à savoir *javazoom*[2]. Le fonctionnement global est assez simple, lorsque le lecteur reçoit un fichier MP3, il le décode en un flux de bits qu'il "écrit" ensuite sur la sortie audio de l'ordinateur. Le lecteur ne permet pas de s'arrêter au milieu d'une lecture et de reprendre par la suite et ne fournit pas un support simple pour avancer/reculer dans un flux. Dans ce mémoire, le lecteur (classe **AdvancedPlayer**) a été étendue par la classe **SongPlayer** afin d'y incorporer la possibilité d'interrompre provisoirement une lecture ("faire pause"). La classe **AdvancedPlayer** fait appel à la classe **Bitstream** qui est en charge de la conversion du fichier MP3 en un flux de bits compatibles avec la sortie audio et à la classe **Decoder** qui elle s'occupe de décoder le fichier MP3.

Les détails techniques du décodage et de la lecture ne faisant pas partie du cadre de ce mémoire, ils ne sont donc pas donnés ici. Il s'agit là d'ailleurs de la raison principale pour l'utilisation d'une telle librairie afin de simplifier le travail. Le lecteur désireux d'en apprendre plus est invité à se rendre sur le site de *javazoom* et de lire la documentation de **JLayer**.

3.3.2 Obtenir des informations d'un fichier MP3 - JID3

JID3[3] est une librairie JAVA sous licence LGPL permettant d'interagir avec les *tags ID3v1 et ID3v2* d'un fichier MP3. Ces tags sont des métadonnées stockées dans un fichier MP3 et qui permettent de renseigner l'utilisateur (ou un logiciel) sur le contenu du fichier. Dans ces tags figurent par exemple l'auteur de la musique, l'album, la pochette de l'album, le genre ou encore le titre de la musique. Comme il a été expliqué à plusieurs reprises, **SmartPlayer** est totalement dépendant de ces tags, il était donc nécessaire de pouvoir y accéder de manière simple et rapide. C'est ce qui a motivé l'utilisation de **JID3** qui permet d'accéder aux tags sans avoir à décortiquer soi-même le fichier MP3 afin de trouver et décoder les informations désirées.

2. www.javazoom.net/javalayer/sources.html
3. http://jid3.blinkenlights.org

3.3.3 Système de log - programmation orientée aspect

Introduction à la programmation orientée aspect

Le système de log utilisé par **SmartPlayer** utilise un paradigme de programmation assez récent (1997 [26]) créé en grande partie par Kiczales, un chercheur de **Palo Alto** en **Californie**. Ce paradigme peut être associé aussi bien à un langage de programmation procédural qu'un langage orienté objet. Dans le cadre de ce mémoire il s'agit d'utiliser l'extension ***AspectJ*** du langage JAVA qui permet donc d'utiliser ce paradigme supplémentaire. **AspectJ** nécessite une étape de précompilation mais le résultat final est du code bytecode tout à fait standard. L'utilisation principale des **aspects** concerne généralement des préoccupations qui ne font pas partie du cœur du programme telle que de l'archivage, de la gestion d'utilisateur, du multithreading ou encore, et c'est ce qui a motivé l'utilisation de ceux-ci, un **système de log**.

Point de vue fonctionnement, il s'agit de modules (appelés aspects) indépendants de l'application principale interagissant avec celle-ci uniquement à travers un modèle de *join points*. Ce dernier détermine à chaque point d'exécution du programme si un ou plusieurs aspects doivent être exécutés et si c'est le cas, le(s)quel(s). Afin de simplifier cette gestion, le système d'aspects permet à l'utilisateur de définir des **pointcuts** au travers d'expression régulière ; ceux-ci permettant de regrouper plusieurs **join points** et de définir les instructions à exécuter lors de tous ceux-ci.

Exemple 3.3.1 (JAVA)**.** *Un pointcut défini comme suit :*

```
execution(* main(..))
```

regroupe les join points de chaque exécution d'une méthode de n'importe quel type de retour, dont le nom est "main" et avec n'importe quel nombre et type d'argument.

AspectJ : la programmation orienté aspect pour JAVA

L'exemple 3.3.1 introduit la syntaxe de définition des pointcuts. Il fait appel au mot clé "execution" mais il en existe d'autres comme **"call"**, **"withincode"** ou encore **"handler"**. Il est également possible de regrouper plusieurs pointcuts en un seul via les opérateurs booléens " !", "&&" et "||". Une fois les pointcuts intéressants définis, le programmeur doit, pour chaque **pointcut**, définir des *advices*. Ces derniers permettent de déterminer à quel moment par rapport au **pointcut** la portion de code à exécuter doit l'être. Parmi ces **advices**, il existe *before*, *after* (que l'on peut coupler avec *returning* s'il est nécessaire que la méthode ait retourné un résultat ou *throwing* dans le cas d'une fin d'exécution générant une exception) ou encore *around* qui permet de remplacer l'exécution normale par un bout de code.

Le lecteur désireux d'obtenir une liste exhaustive des pointcuts et advices existants ou d'autres informations précises à ce sujet devrait se tourner vers la documentation officielle de **AspectJ** [4].

Exemple 3.3.2. *S'il est désirable d'exécuter une portion de code particulière dès qu'une exception est lancée dans le constructeur d'une classe spécifique. Pour ce faire il faut définir le pointcut comme suit :*

```
pointcut constructorFail():
withincode (DesiredClass.new(..))
&& handler(Exception);
```

- *pointcut permet de déclarer un pointcut avec le nom qui le suit (constructorFail ici),*
- *withincode() désigne tous les **join points** existant dans un constructeur de DesiredClass,*
- *handler() désigne tous les **join points** correspondant à des gestionnaires d'exceptions, c'est-à-dire tous les blocs "catch",*
- *le "&&" permet de faire l'intersection des deux spécifications.*

*Le **pointcut** désigne donc tous les **join points** correspondant à des blocs "catch" dans un constructeur de la classe DesiredClass.*
Les advices suivants pourraient être implémentés :

```
before() : constructorFail()
{
    System.out.println("Exception qui va être traitée");
}
after() : constructorFail()
{
    System.out.println("Exception qui a été traitée");
}
```

Ce dernier advice est erroné (il ne sera jamais exécuté) car il est impossible d'exécuter un advice après qu'une exception ait été traitée vu que le fil d'exécution est interrompu.

Avantages. Le principal avantage des aspects est qu'ils permettent de réduire voire supprimer le couplage entre les fonctionnalités vitales à l'application et les fonctionnalités supplémentaires telles qu'un système de log. De cet avantage découlent :

- **une maintenance aisée** car les aspects peuvent être modifiés sans que le code principal ne soit impacté (et vice-versa),
- **une meilleure réutilisation** car les aspects peuvent être redéployés sur un autre environnement ou sur un environnement qui a évolué,
- **une meilleure qualité du code** car simplifié, modulaire à souhait.

4. http://www.eclipse.org/aspectj/doc/released/progguide/index.html

Inconvénients. Le tisseur (le pré-compilateur) est en fait un générateur de code et ce dernier peut être difficile à analyser pour le débogage. Il existe cependant des outils sophistiqués qui permettent de passer de façon transparente du code d'une classe au code d'un aspect (le plug-in eclipse utilisé pour ce mémoire est très facile d'utilisation). Le modèle d'aspects greffé peut peut-être également réduire les performances de l'application mais rien de bien probant.

Système de log

Le système de log utilisé fait appel à la classe **Logger.java** permettant l'écriture dans des fichiers de logs (un fichier par jour) ainsi qu'une interface **Dumpable** permettant de définir ce qui est à logger. Chaque classe ou groupe de classes servant à remplir une fonction contenant des méthodes devant être loggées est associé(e) à un fichier d'aspects dont le nom est proche (par exemple *MP3StorageModel* $\rightarrow$ *StorageModelAspects*) qui s'occupe de logger tout ce qu'il faut. Chacun de ces aspects étend la classe **LoggingAspect** (cf figure 3.12) qui contient les mécanismes de base permettant de spécifier si les logs doivent contenir des détails très précis ou ne contenir que le principal.

Chaque ligne de log est composée
- d'un minuteur dont la précision va jusqu'aux millisecondes,
- d'un contexte à spécifier
  (souvent le nom de la classe et/ou de la méthode),
- d'un message expliquant l'événement survenu,
- un nombre indéfini de *dump* de variables (impression du résultat de la méthode dump() requise par l'interface **Dumpable**).

```
16:45:48:343 -- [DatabaseManager] Following query is going to be executed.
   0 : SELECT * FROM similarities WHERE idA = '1' AND idB = '2';
```

FIGURE 3.11 – Exemple de ligne de log

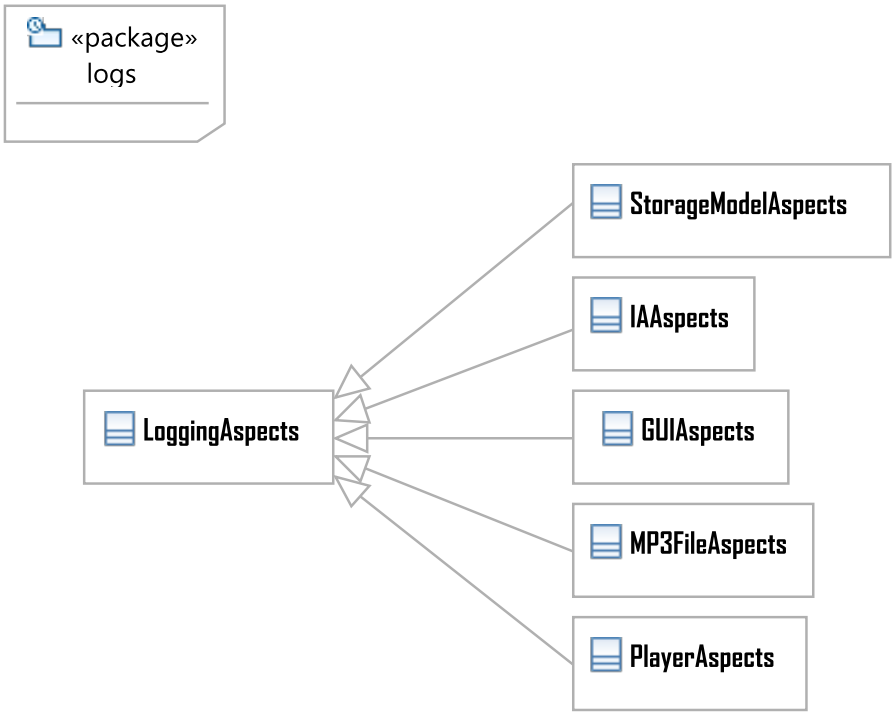


FIGURE 3.12 – Diagramme de classe des aspects utilisés pour le système de log

3.3.4 Système de configuration

Le logiciel intègre un système de configuration extensible via l'utilisation d'un fichier de configuration et de la classe **ConfigManager** faisant le lien entre celui-ci et une table de hashage disponible de manière statique. Chaque ligne du fichier est un couple *clé :valeur* où la clé est une chaîne de caractères et la valeur une représentation sous la forme d'une chaîne de caractères d'une instance de la classe **SettingValue**. Cette classe permet de manière basique de représenter une valeur de type entière, booléenne et chaîne de caractères mais est facilement extensible.

La valeur de configuration la plus importante est la valeur de *feedback* qui permet à l'utilisateur d'activer ou non le "Mood sensing". Le fichier de configuration contient également et entre autres le dossier par défaut où chercher les musiques ou encore la taille de la fenêtre principale. La fenêtre utilisée par l'application pour afficher et permettre l'édition de ces valeurs de configuration est implémentée de sorte à créer autant de panels de modification qu'il y a de valeurs de configurations stockées dans le fichier de configuration. Ceci, combiné à l'extensibilité de la classe **SettingValue** permet à ce système de configuration d'accueillir de nouvelles valeurs de configuration de manière pratiquement transparente (il reste à répercuter ces ajouts dans les classes devant utiliser cette valeur).

3.3.5 Apparence et fonctionnement de la GUI

Lorsque l'utilisateur lance l'application, la fenêtre principale s'affiche et une musique dont l'appréciation est parmi les plus grandes est sélectionnée comme première musique. L'utilisateur a le choix de la lire directement ou d'ouvrir une autre musique. La liste de lecture est remplie au fur et à mesure qu'une musique est lue (ou passée). Cette fenêtre sommaire est illustrée par la figure 3.13.

FIGURE 3.13 – Fenêtre principale de l'application

Cette fenêtre intègre également deux menus : *File* et *Edit*.
Le menu **"File"** est illustré dans la figure 3.14 et il regroupe toutes les actions possibles à partir de fichiers. Les actions possibles sont :

- **Ouvrir une musique** (Open...) en la sélectionnant directement dans l'arborescence de son ordinateur. Si la musique est inconnue du lecteur, il la lit et en arrière plan l'ajoute à la librairie tout en effectuant la tâche de clustering afin de déterminer les similarités de cette musique avec celles déjà connues (voir figure 3.15).
- **Ajouter une musique** à la bibliothèque mais sans la lire. Les actions prises par le lecteur sont alors pratiquement identiques que celles effectuées lors de l'ouverture d'un fichier mis à part qu'un pop-up d'erreur s'affiche lorsque la musique se trouve déjà dans la bibliothèque et la musique ajoutée n'est pas lue.
- **Quitter l'application**.

FIGURE 3.14 – Menu “File”

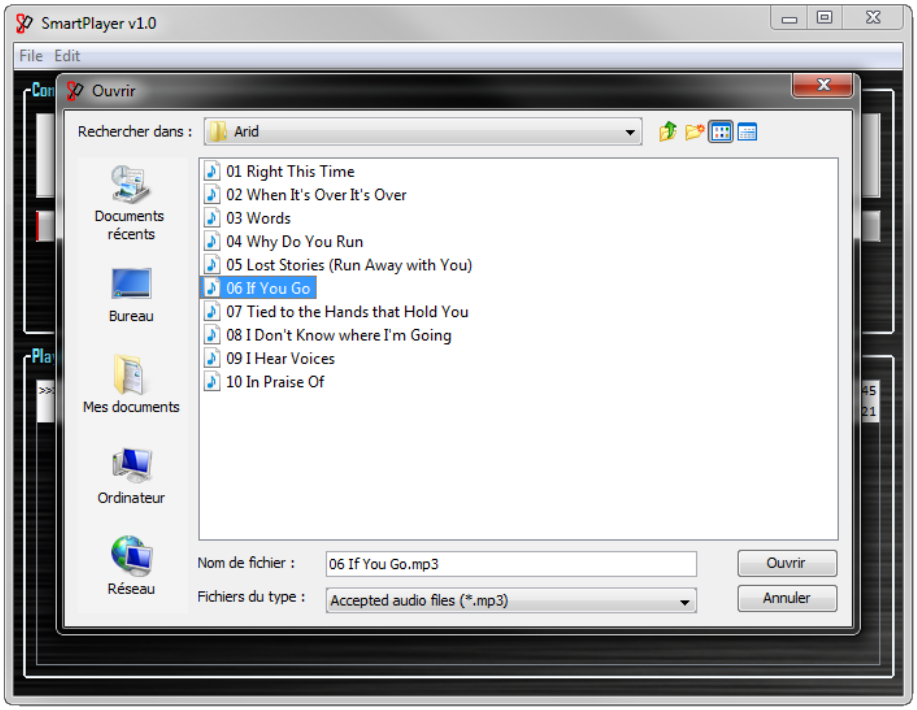


FIGURE 3.15 – Fonctionnalité “Ouvrir une musique”(Open...)

FIGURE 3.16 – Menu “Edit”

Le menu **“Edit”** est illustré dans la figure 3.16 et contient toutes les actions permettant d’éditer le comportement du programme. Il permet donc de :

- **Modifier la configuration du logiciel** (via le fichier de configuration), une fenêtre s’affiche avec les différentes valeurs de configuration et un moyen de les modifier (cf figure 3.17). L’utilisateur a alors le choix de valider ses changements ou de les annuler. En cas de validation, les modifications sont répercutées sur le fichier de configuration. L’utilisateur peut (dés)activer le feedback (“Mood Sensing”) via ce menu.
- **Vider la liste de lecture**, l’utilisateur devant ouvrir une musique de son choix pour continuer l’utilisation. Si le lecteur était en pleine lecture d’une musique, il se stoppe et supprime tout son historique de musiques déjà lues.

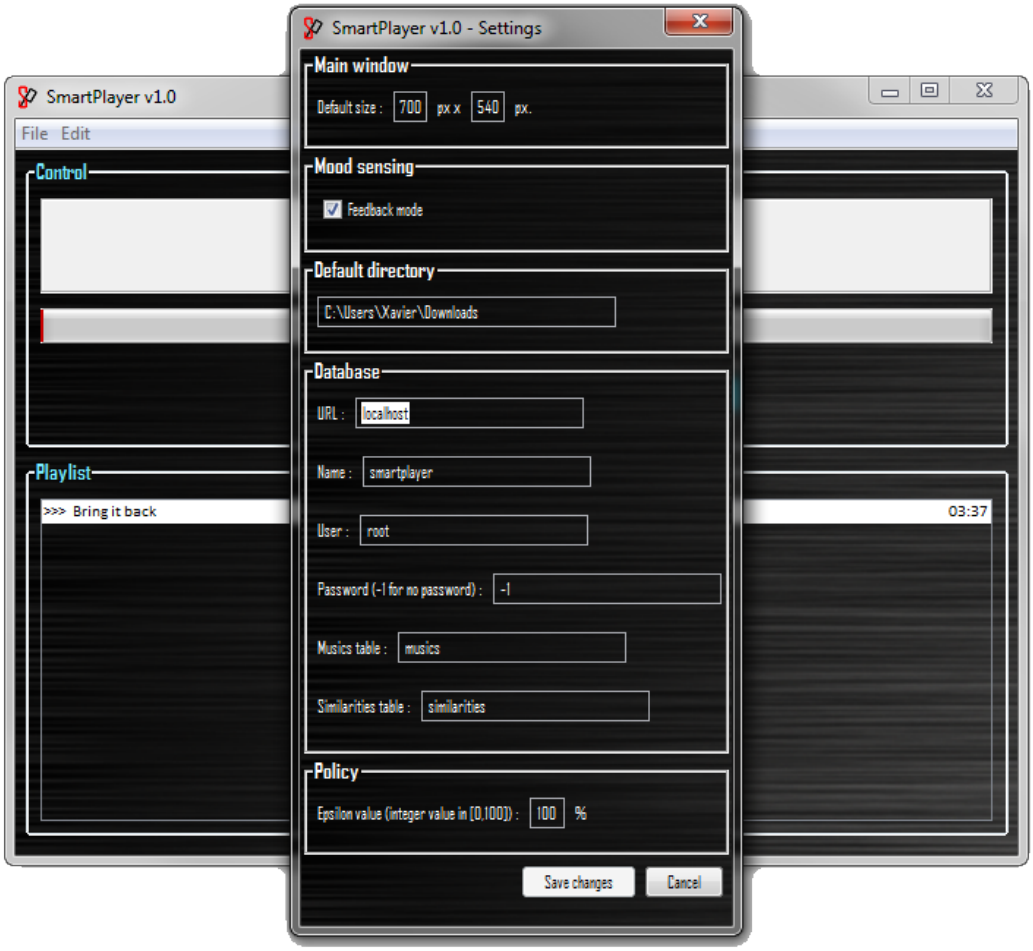


FIGURE 3.17 – Fenêtre d'édition de la configuration

Lorsque l'utilisateur passe une musique ou que celle-ci est terminée, **SmartPlayer** détermine la prochaine chanson à lire et lance sa lecture. Si le "Mood Sensing" est activé, un pop-up apparaît alors demandant à l'utilisateur si le choix effectué rencontre ses critères ou pas. L'utilisateur dispose alors de 5 secondes avant que le pop-up ne se ferme automatiquement et détermine son contentement comme "Inconnu". Si l'utilisateur est content, **SmartPlayer** reçoit une récompense supplémentaire de $+1$ tandis que s'il ne l'est pas, ce dernier reçoit une pénalité de -1. Lorsque le contentement est évalué comme "inconnu" ou "indifférent", cette récompenses supplémentaire est nulle. L'utilisateur a également la possibilité de cocher une option dans ce pop-up spécifiant à l'application qu'il ne veut plus être importuné par cette fonctionnalité. Cette fonctionnalité est illustrée en action dans la figure 3.18.

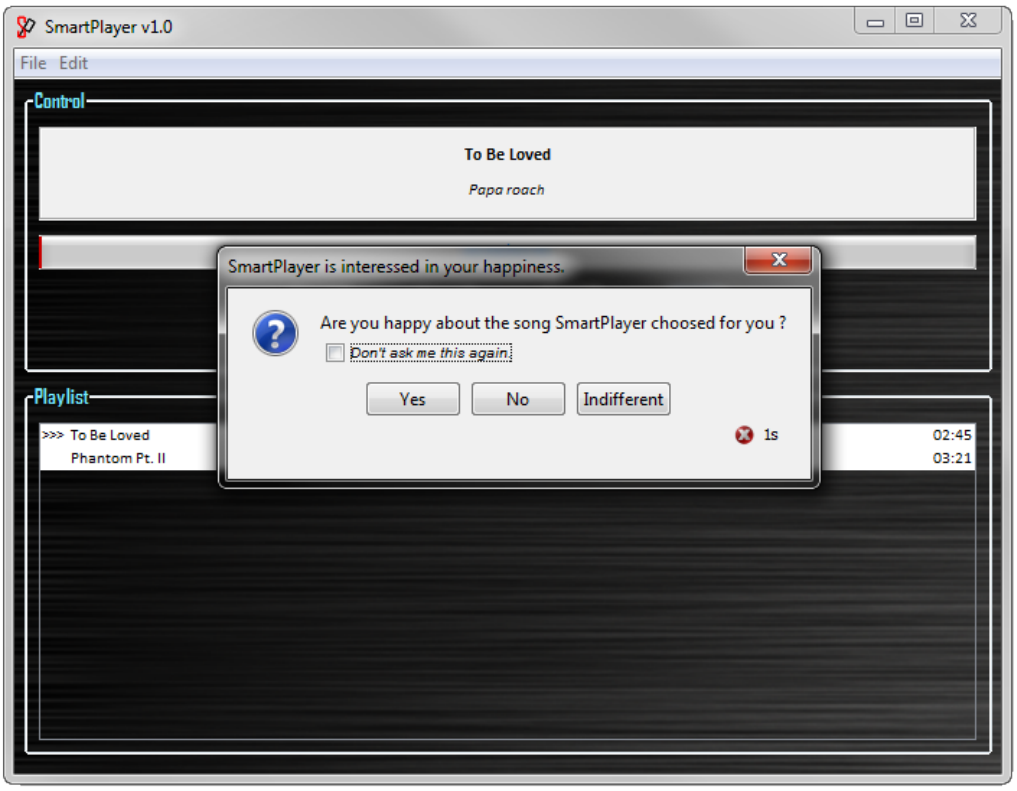


FIGURE 3.18 – Mood Sensing en action

3.4 Difficultés

Cette section vise à décrire les difficultés éprouvées lors de l'accomplissement de ce mémoire. Ces difficultés ont été pour la plupart concentrées sur l'implémentation de l'intelligence artificielle car il s'agit de la partie la plus complexe du logiciel. Les autres quant à elles se regroupent sous le souci de performance du programme, l'interface graphique ainsi que de l'obtention d'informations pour l'état de l'art.

3.4.1 Implémentation de l'intelligence artificielle

La partie la plus compliquée est la fonction de valeur. En effet, il a été compliqué de comprendre et d'implémenter la manière dont ces valeurs sont évaluées. Les valeurs étant ces similarités stockées par le logiciel et qui servent également de récompenses, cela a souvent prêté à confusion. La fonction de valeur a donc été modifiée pour effectuer un calcul préventif sur un nombre d'étapes fixé afin de pouvoir séparer les notions de valeur et de récompense.

Le fait que l'intelligence artificielle devait à la fois choisir la musique suivante et modifier les similarités prêtait également à confusion car cela laissait penser que l'agent pouvait modifier ses récompenses (ce qui est contraire à la définition de la fonction de récompense). Cependant, lorsque l'on s'accroche à la définition formelle, la fonction de récompense n'est pas modifiable directement et le fait de mettre toutes les similarités à 1 n'amènerait pas le maximum de récompense et ce, grâce au **Mood Sensing**. En effet, celui-ci, si l'utilisateur n'est pas content, même si la similarité est de 1, apportera une récompense moins élevée que si l'utilisateur était content.

3.4.2 Informations

La plupart des logiciels de ce type qui existent sont des logiciels payants et non open-source. Ceci a pour conséquence que pour obtenir de l'information sur leur fonctionnement et pouvoir tester ces logiciels pleinement relève de l'épreuve car tout ceci est gardé bien jalousement (ce qui est compréhensible). C'est pour cela que la plupart des informations récoltées sont basées sur des articles qui semblent être fiables mais dont la véracité n'a évidemment pas pu être totalement vérifiée.

3.4.3 Performance

La lecture de fichiers MP3 via un thread synchronisé avec un sémaphore permettant d'interrompre la lecture et de la reprendre n'est probablement pas la meilleure façon de faire. De plus, l'implémentation donnée du modèle utilisé pour stocker les similarités est une base de données et peut-être que de stocker la demi-matrice de similarité sous cette forme n'est pas la manière la plus optimale. Il existe probablement d'autres notions qui, liées au deux précédemment citées, font que le logiciel n'est sans doute pas le plus performant qui soit et peut par moment prendre quelques instants avant de réagir à l'action demandée par l'utilisateur. Le logiciel n'est pas développé dans un souci de performance mais ceci est toujours une préoccupation importante qui soulève quelques questions et quelques difficultés.

3.4.4 L'interface graphique

Il a été particulièrement fastidieux d'implémenter le **Mood Sensing** via le pop-up modal qui se ferme automatiquement après 5 secondes. En effet, la librairie JAVA n'est pas très complète à ce sujet et trouver comment ajouter une *"checkbox"* dans un objet de la classe **JDialog** ou **JOptionPane** et en récupérer l'état lors de la fermeture du pop-up fut une tâche beaucoup plus longue que prévue.

Il n'a pas été évident non plus de synchroniser le lecteur avec tous les éléments de l'interface graphique. Par exemple, que la barre de progression s'arrête dès que l'utilisateur clique sur pause, et qu'elle reparte dès qu'il clique sur "play" ou que la liste de lecture s'étende et que le titre et l'artiste de la musique courante changent instantanément lorsque l'utilisateur clique sur "next".

3.5 Tests & Résultats

Cette section a pour but de valider le logiciel, de prouver qu'il fonctionne. Dans la littérature [19], pour prouver qu'un agent d'apprentissage par renforcement apprend réellement, sa récompense moyenne est analysée et celle-ci doit augmenter au cours du temps. Dans le cas de **SmartPlayer**, il peut également être judicieux de vérifier que les similarités et appréciations convergent bien vers des valeurs qui tendent à être stables. Finalement, l'application étant multithreadée, il est également intéressant d'effectuer un monitoring du nombre de threads utilisés par le logiciel. Pour tous ces tests, le même échantillon de musiques est utilisé.

Cet échantillon se veut être le plus représentatif possible et comporte des musiques de genre, d'artistes et de titres divers. Il y a également quelques musiques dont l'artiste est identique ou dont le titre est identique ainsi que quelques musiques dont le genre, l'artiste et/ou le titre ne sont pas renseignés. Cet échantillon est détaillé dans l'annexe A.

Le logiciel dépendant très fortement des goûts de l'utilisateur, de tels tests, pour être totalement convaincants, devraient être appuyés par une analyse statistique des données recueillies lors de l'utilisation du logiciel par un échantillon représentatif de la population ciblée. Cependant, il est compliqué d'effectuer une telle analyse dans le cadre d'un mémoire. Les tests recueillis ici sont donc effectués majoritairement par le programmeur qui a pris bien soin de fixer les goûts des utilisateurs dont il jouait le rôle avant de commencer ces tests.

3.5.1 Test des récompenses

Deux tests ont été effectués : lorsque le **Mood Sensing** est activé et lorsque cette fonctionnalité ne l'est pas. En effet, cette fonctionnalité a pour effet d'augmenter ou de diminuer la récompense de 1 et a donc un impact non négligeable sur la récompense moyenne reçue. Pour chacun de ces deux tests, trois différentes analyses sont menées :

1. **Evolution de la récompense moyenne obtenue par *exécution***, cette valeur étant évaluée sur 10 exécutions. Une exécution est, ici, considérée comme la lecture à la suite de 20 musiques de l'échantillon, pas plus, pas moins. Cette analyse est la plus importante car elle permet d'apprécier l'efficacité de l'apprentissage.

2. **Evolution des récompenses moyennes obtenues sur une exécution**, ces valeurs étant des moyennes obtenues sur 10 exécutions également. Ces valeurs représentent en fait les récompenses obtenues le long d'une exécution moyenne, il est donc possible de consulter la récompense moyenne obtenue par l'agent lors du passage de la $10^{\text{ème}}$ à la $11^{\text{ème}}$ musique. Cette analyse est surtout là à titre informatif et peut permettre de montrer que ce n'est pas les plus grosses récompenses qui amènent la meilleure somme totale de récompenses.
3. **Evolution des récompenses obtenues sur une exécution** qui est un complément de l'analyse précédente.

Dans ces trois analyses, les moyennes sont obtenues sur 10 exécutions de l'application avec une valeur ϵ qui est diminuée progressivement. Pour rappel l'application fait appel à la politique ϵ-greedy qui consiste à agir de manière aléatoire avec une probabilité égale à ϵ et de manière gloutonne avec une probabilité égale à $1 - \epsilon$ ($\epsilon \in [0, 1]$). Ainsi, les quatre premières exécutions sont exécutés avec $\epsilon = 0.2$, les quatre suivantes avec $\epsilon = 0.1$ et les deux derniers avec $\epsilon = 0$. L'idée derrière ce procédé est que l'agent doit explorer de moins en moins au fur et à mesure qu'il "connaît" les musiques et les goûts de l'utilisateur pour celles-ci. Bien sûr, en cas d'ajout de nouvelles musiques, l'agent se doit d'adapter ϵ afin d'explorer à nouveau.

Sans Mood Sensing

La figure 3.19 montre l'évolution de la récompense moyenne obtenue par l'agent lors d'une exécution sans Mood Sensing (moyenne de 10 exécutions). Dans cette analyse, l'évolution de l'agent n'utilisant pas la fonction de valeur et donc sans se projeter dans le futur est également montrée. Il s'agit d'une évolution totalement gloutonne ne se basant que sur la récompense directement reçue. Ce graphique permet en outre de montrer que cette approche est la pire de celles qui sont illustrées.

Comme il était attendu, la meilleure politique est celle utilisée par l'agent lors des deux derniers runs. Il pourrait être intriguant que lors de la sélection des dernières musiques, la politique finale soit celle qui obtient la récompense moyenne la plus basse. Cependant, ceci est très facilement explicable. En effet, si l'utilisateur n'aime environ que 10 musiques de l'échantillon, l'agent finit par les jouer dans les 10 premières places et par la suite il place les "moins pires". Le problème est que ces dernières ont été désapprouvées par le passé et donc leur appréciation et même la similarité entre elles est (beaucoup) plus faible que lors des premiers runs.

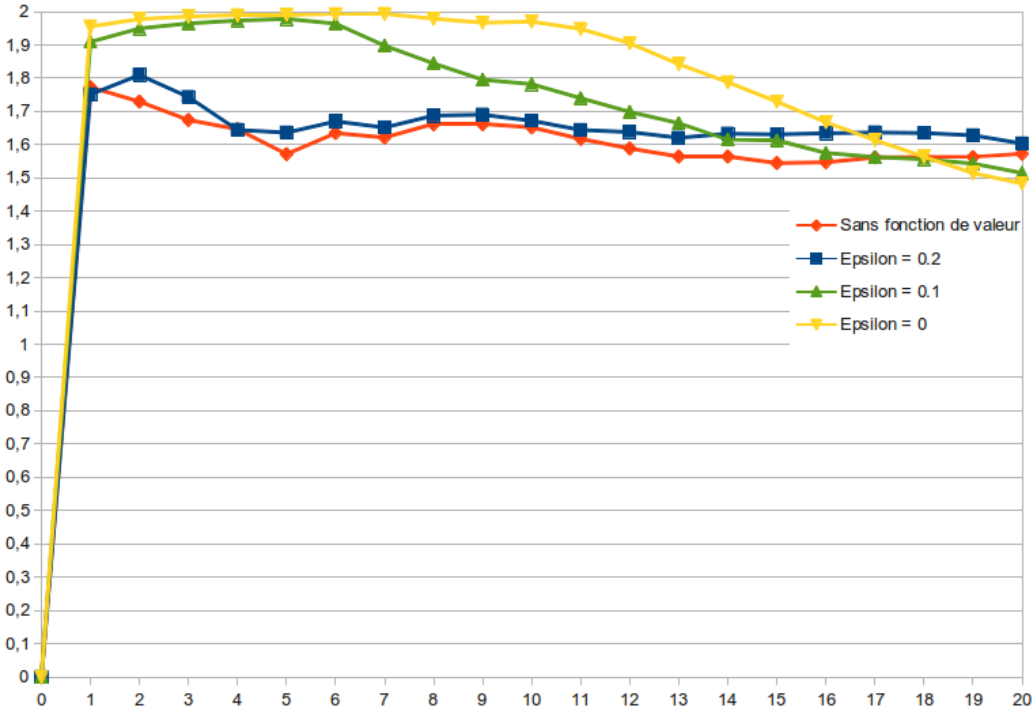


FIGURE 3.19 – Récompense moyenne obtenue sur une exécution

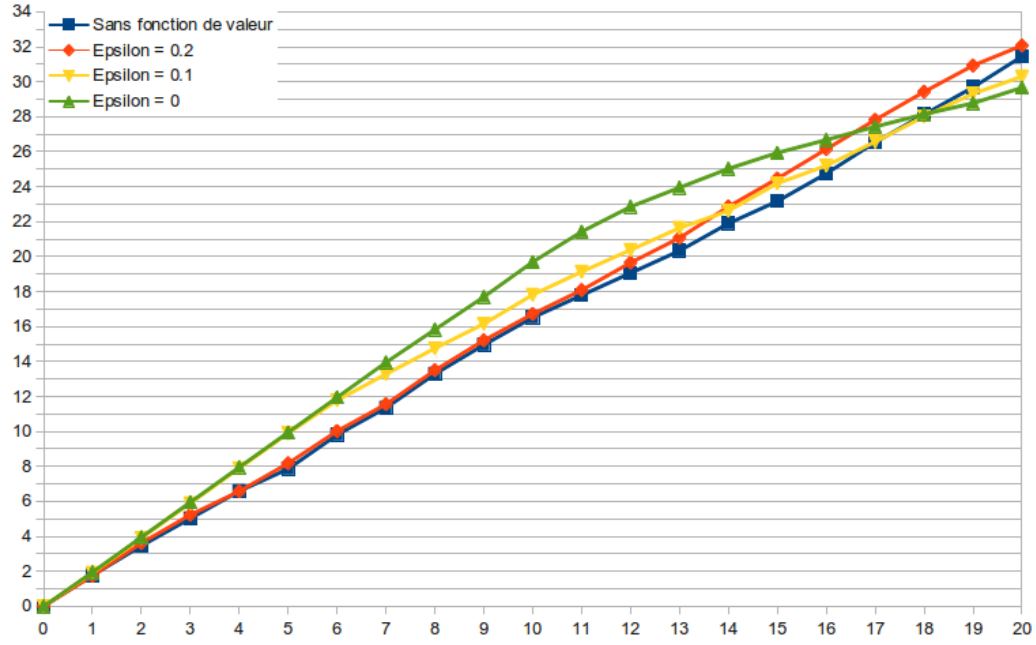


FIGURE 3.20 – Récompense obtenue sur un run

La figure 3.20 montre l'évolution de la récompense obtenue par l'agent lors d'une exécution sans Mood Sensing (moyenne de 10 exécutions). Celle-ci permet les mêmes constatations que la figure 3.19, à savoir que la politique finale est la meilleure (et elle ne l'est que si l'agent a exploré au préalable grâce aux valeurs plus grandes de ϵ afin d'effectuer son apprentissage).

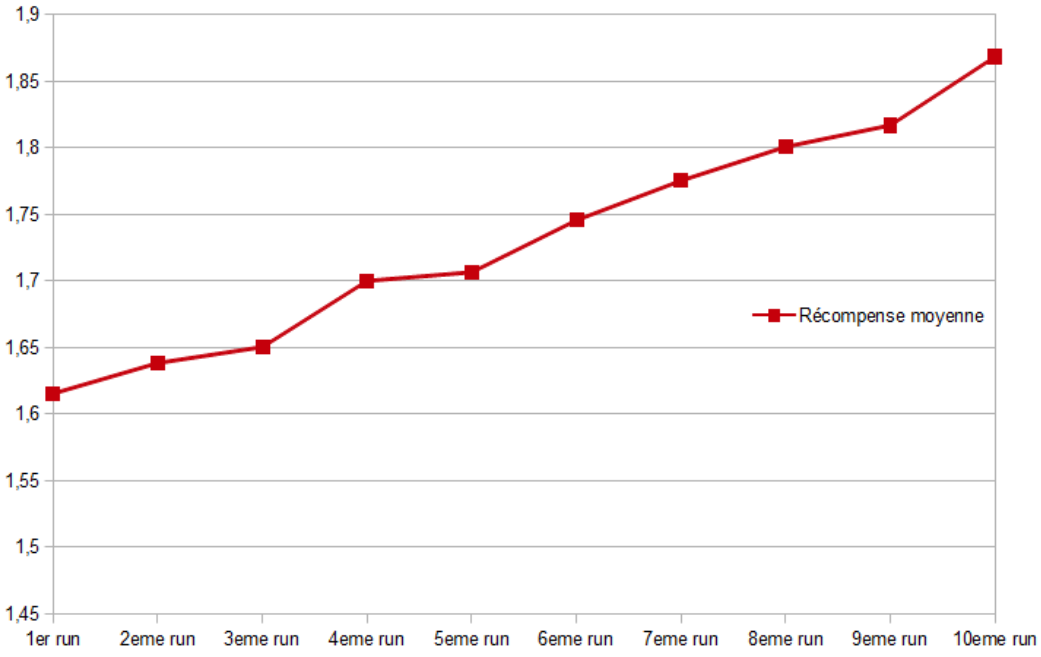


FIGURE 3.21 – Récompense moyenne obtenue au terme de 10 exécutions

La figure 3.21 est la plus importante. Cette figure illustre l'évolution de la récompense moyenne obtenue par exécution du programme. Cette évolution étant une croissance, cette récompense moyenne ne cesse d'augmenter au fur et à mesure que l'utilisateur exécute le programme. Il s'agit donc là d'une preuve empirique que l'agent **apprend** et les récompenses qu'il collecte se rapprochent toutes de la récompense maximale qui est de 2 sans Mood Sensing.

Avec Mood Sensing

Les mêmes analyses ont été effectuées et les résultats sont pratiquement identiques. Ces résultats sont reproduits dans les figures 3.22, 3.23 et 3.24. Ce que le Mood Sensing apporte est un **apprentissage plus rapide et plus efficace**. En effet, dans ce cas-ci les récompenses sur une exécution sont toujours supérieures, même en fin de l'exécution, pour la politique avec $\epsilon = 0$. Cette dernière correspondant à la politique finale de l'agent, politique qui exploite au maximum les connaissances de l'agent.

Une autre différence ici est que les récompenses peuvent être négatives mais également atteindre 3 ce qui n'a pas un impact très frappant sur les graphiques mais a un impact non négligeable sur l'agent. En effet, en cas d'une récompense r reçue normalement, si l'utilisateur n'est pas content l'agent obtient $r - 1$ et si celui-ci l'est, ce dernier obtient $r + 1$ soit une différence de 2 unités.
Ces 2 unités représentent également la récompense maximale sans Mood Sensing. Cette différence n'est donc pas négligeable et pousse l'agent "à ne vouloir absolument pas" mécontenter l'utilisateur afin de ne pas perdre de points.

La figure 3.23 montre que les 12 premières musiques rapportent à l'agent des récompenses pratiquement égales à 3, ces récompenses diminuant par la suite. Ceci illustre une nouvelle fois le phénomène observé plus tôt : l'utilisateur apprécie une dizaine de musiques de l'échantillon et ces musiques sont placées en avant par l'agent. Par la suite l'agent finit par ajouter les musiques rapportant une somme globale de récompenses maximale bien que ces musiques ne soient pas très appréciées par l'utilisateur.

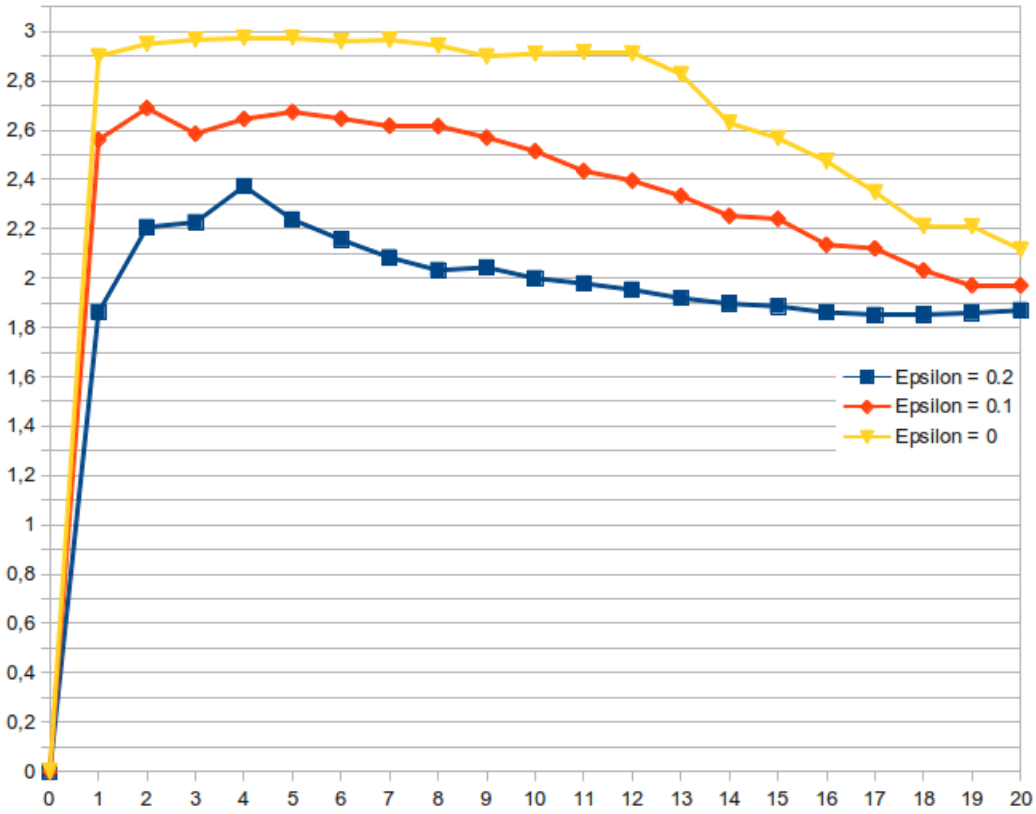


FIGURE 3.22 – Récompense moyenne obtenue sur une exécution

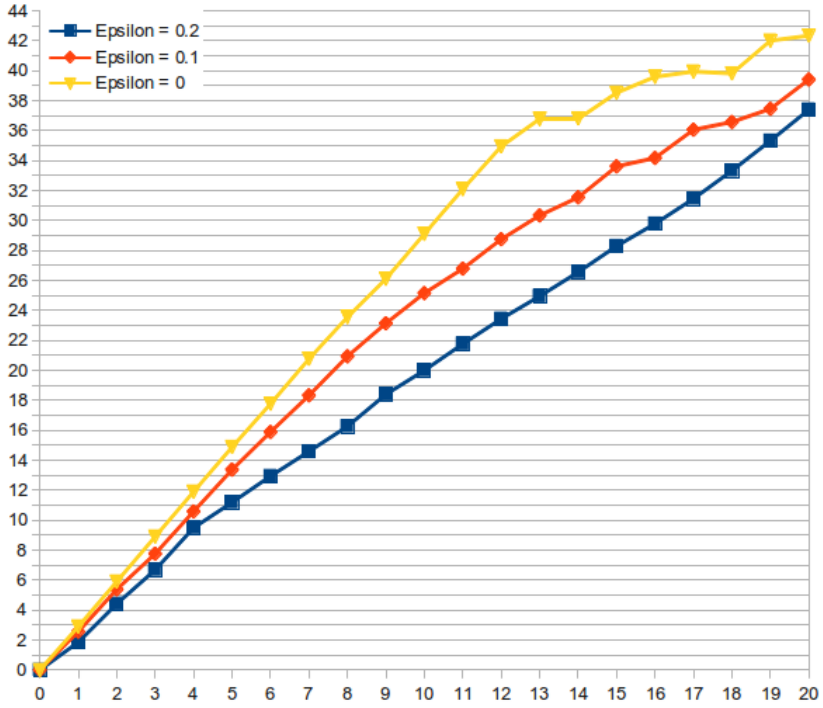


FIGURE 3.23 – Récompense obtenue sur une exécution

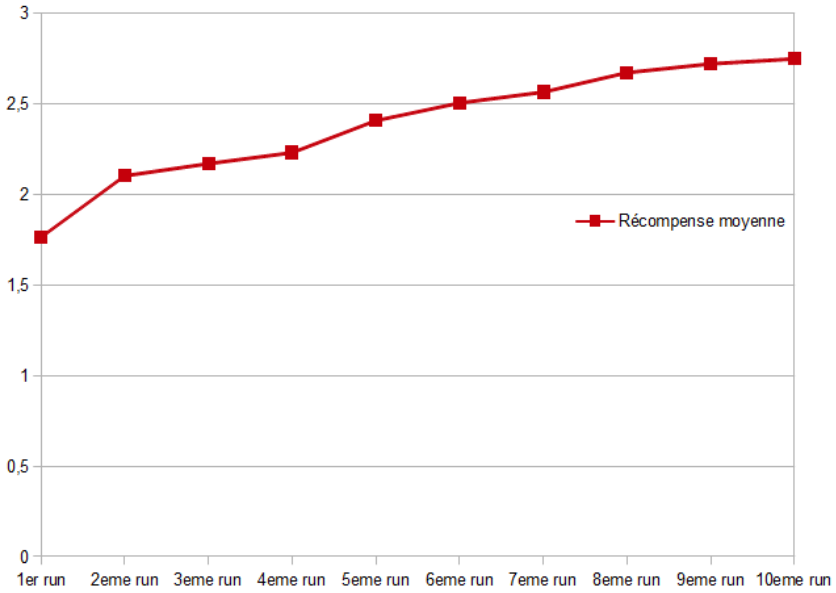


FIGURE 3.24 – Récompense moyenne obtenue au terme de 10 exécutions

3.5.2 Test des similarités et appréciations

Ces tests permettent de mettre en lumière l'exploration de l'agent. En effet, les similarités et les appréciations sont les valeurs que l'agent apprend et donc modifie au cours du temps. Si toutes les similarités sont modifiées à la fin de plusieurs exécutions, c'est que l'agent a exploré au moins une fois chaque couple de musiques possible. Cette situation est très peu probable car les similarités initiales pousseront l'agent à choisir certaines chansons plutôt que d'autres. Il est cependant possible de la favoriser en plaçant ϵ à une valeur élevée pour les premières exécutions. Ceci ayant pour conséquence d'augmenter le taux d'exploration de l'agent.

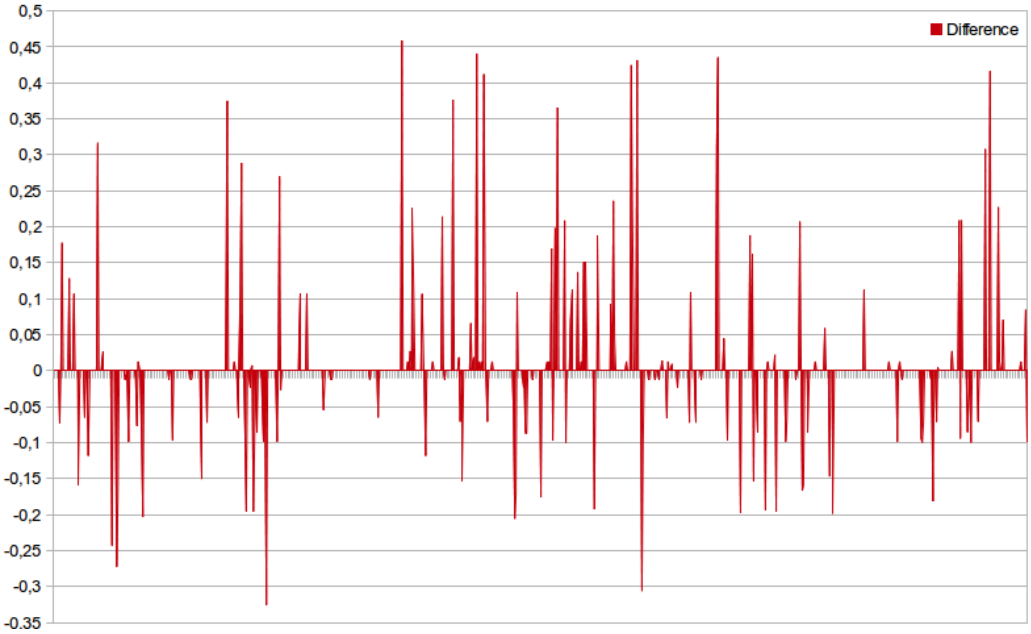


FIGURE 3.25 – Différences entre les similarités finales et les similarités initiales

La figure 3.25 représente les différences entre les similarités finales et les similarités initiales et la figure 3.26 établit pratiquement la même relation pour les appréciations. La différence étant prise dans l'autre sens vu que l'appréciation initiale de chaque musique est considérée comme égale à 1 (le maximum) afin que toutes nouvelles musiques soient favorisées à la lecture. Ces graphiques montrent que ces valeurs ne varient pas de manière uniforme ce qui est normal et souhaité car l'utilisateur ne peut aimer toutes les chansons ni les aimer autant les unes les autres.

La figure 3.25 montre surtout que certaines similarités ne sont jamais modifiées. Ceci est compréhensible car l'agent ne peut proposer à l'utilisateur toutes les combinaisons de musiques possibles. Cela peut-être dommageable car il se peut que deux musiques très proches l'une de l'autre ne soient jamais jouées l'une après l'autre et l'utilisateur ne pourra donc jamais spécifier à l'agent qu'il est content d'un tel choix. Cependant cette situation est très peu probable car il faudrait que la similarité initiale de ces deux musiques ne soient pas conséquente et que la politique n'explore que très peu.

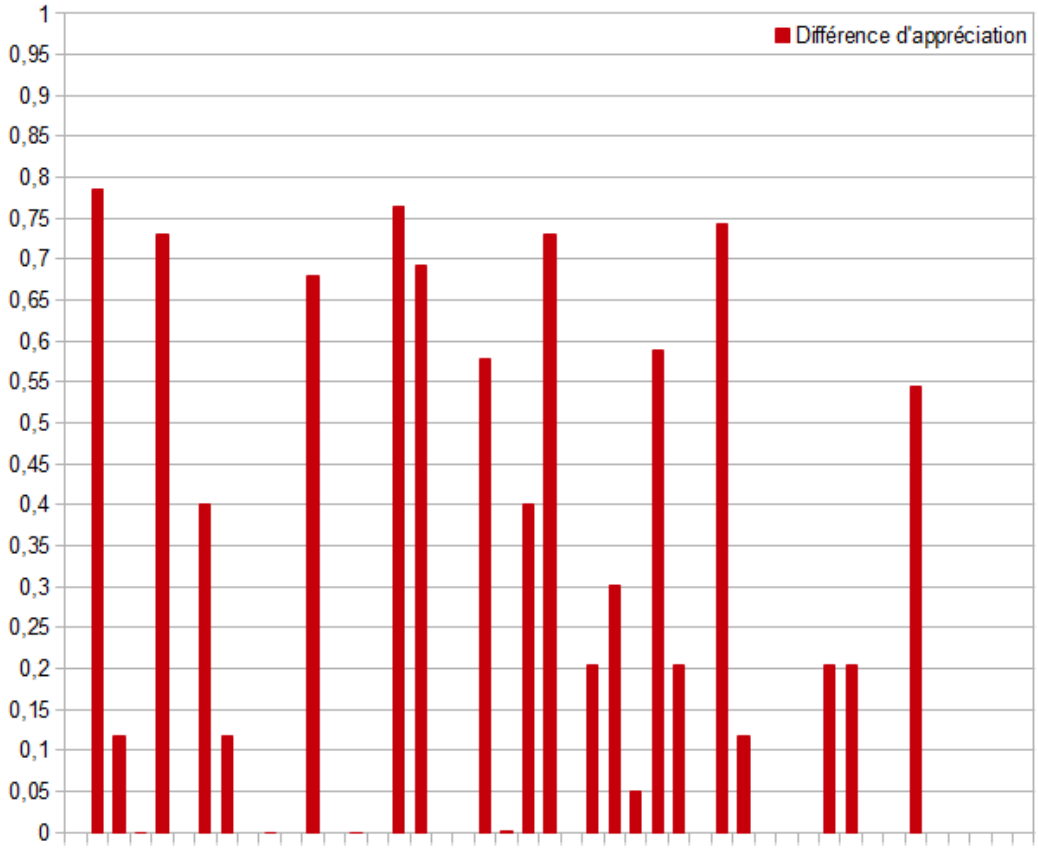


FIGURE 3.26 – Différences entre les appréciations initiales et finales

Sur la figure 3.26, le même phénomène est constaté bien qu'il soit un peu différent. En effet, ici, toutes les musiques ont été jouées au moins une fois, seulement l'utilisateur est resté fidèle à certaines musiques et l'appréciation de celles-ci n'a donc pas été modifiée (vu que la fonction d'augmentation f possède un point fixe en 1, c'est-à-dire que $f(1) = 1$).

3.5.3 Monitoring des threads

La figure 3.27 montre l'évolution du nombre de threads en fonction du temps d'exécution (en secondes) de l'application. Le nombre de threads, de manière générale, se maintient à 13 en comptant les 9 threads initialisés par JAVA. Les quatre threads utilisés par l'application en elle-même se résument à :

- **le thread "Player"** qui lit les musiques,
- **le thread "AI"** qui représente l'agent prenant les décisions,
- **le(s) thread(s) "AWT"** qui représentent les composants graphiques.

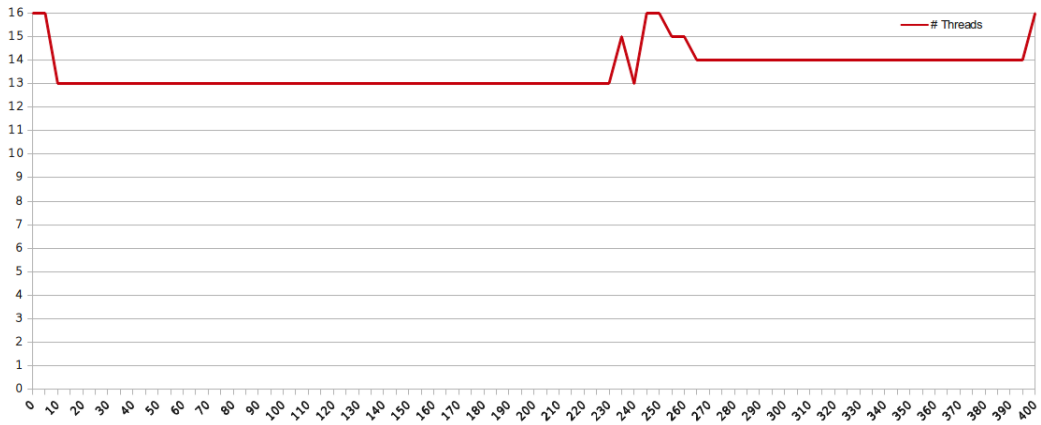


FIGURE 3.27 – Nombre de threads en cours lors d'une exécution

Le scénario retranscrit montre l'influence d'un changement de musique à la $230^{\text{ème}}$ seconde, ceci lançant un thread permettant d'effectuer le Mood Sensing (JAVA lançant également un thread pour afficher la fenêtre de dialogue). Le scénario se poursuit ensuite par un ajout d'une nouvelle musique aux alentours de la $240^{\text{ème}}$ seconde d'exécution. Un thread **"Song adder"** est alors initialisé et exécuté. JAVA lance également plusieurs threads permettant de faire tourner les composants graphiques permettant les choix de fichiers. Un de ces threads continue ensuite son existence en restant inactif, ceci expliquant pourquoi le nombre de threads reste à 14 par la suite. Le pic final correspond à l'action **stop** ainsi qu'à la fermeture du programme, JAVA faisant appel à deux threads pour effectuer ce travail.

Conclusion

Ecouter de la musique est une activité très répandue de nos jours. Cette écoute peut se faire entre autres en utilisant des logiciels de lecture de musiques comme, par exemple, **Apple Genius**, **Jukefox** ou encore **last.fm**. Ces logiciels proposent à l'utilisateur une expérience unique au travers de fonctionnalités telles que la création de listes de lecture intelligentes ou encore la proposition de musiques que celui-ci est susceptible d'aimer. Ils basent pour la plupart leur fonctionnement sur les musiques possédées ou écoutées et non sur l'utilisateur en lui-même. Ces logiciels sont dits 'intelligents" car pour implémenter ces fonctionnalités ils utilisent l'apprentissage automatique qui est un concept de l'intelligence artificielle.

Dans ce mémoire, le développement de **SmartPlayer** a été présenté. Il s'agit d'un logiciel de lecture de musiques dont le fonctionnement se base principalement sur les actions de l'utilisateur. Sa fonctionnalité principale est d'enchaîner des musiques qui plaisent à l'utilisateur de manière aléatoire tout en évitant au plus possible celles que celui-ci n'apprécie pas. Il utilise pour ce faire un système basé sur deux types de coefficients : les similarités entre musiques et l'appréciation d'une musique. Ces coefficients sont déterminés initialement par un apprentissage non-supervisé sous la forme d'un clustering. Ils sont ensuite modifiés à chaque utilisation de l'application grâce à un apprentissage par renforcement dont la politique se base sur la politique ϵ-greedy. Cette politique simple consiste à agir de manière gloutonne avec une probabilité de $1 - \epsilon$ et de manière aléatoire dans tous les autres cas. Un exemple de renforcement utilisé est de réduire la valeur d'appréciation de musiques qui ne sont pas écoutées jusqu'au bout. Le renforcement est optionnellement appuyé par du **"Mood Sensing"** qui consiste à demander à l'utilisateur s'il est content des choix effectués par l'agent, les réponses de l'utilisateur ayant un impact non-négligeable sur l'apprentissage.

L'implémentation et les tests effectués sur ce lecteur ont permis de montrer qu'un tel logiciel d'apprentissage est non seulement possible mais peut également inclure un bon nombre de fonctionnalités différentes. Ces tests pourraient être encore plus probants s'ils étaient accompagnés par un étude statistique, qui pourrait constitué une perspective. La manière de fonctionnement de cette application pourrait être modifiée en explorant d'autres espaces d'hypothèses que celui choisi. Ainsi un réseau bayésien ou des réseaux de neurones pourraient être utilisés.

L'application pourrait également être étendue, en intégrant de nouveaux éléments. Parmi ceux-ci, la prise en charge d'autres formats musicaux comme les formats WMA ou FLAC. Une autre extension possible serait de remplir les tags IDv3 des fichiers de musiques dont ceux-ci sont manquants sur base des similarités avec d'autres musiques dont les tags sont définis. L'application pourrait également s'inspirer d'un des systèmes les plus utilisés dans le domaine à l'heure actuelle, à savoir un système de tags sociaux permettant de tagger les musiques avec des mots qui lui sont associés. Par exemple, une musique romantique pourrait être taggée par les mots "calme" et "amour" mais aussi par "triste" ou tout autre mot que l'utilisateur juge pertinent. Finalement, et c'est sans doute une amélioration des plus intéressantes, l'application pourrait se baser sur les informations provenant de l'analyse du signal d'une musique comme l'amplitude maximale ou le tempo afin d'affiner son clustering permettant de faire une estimation initiale et théorique de la similarité entre deux musiques.

Finalement, il a été également exhibé qu'un tel logiciel dépend fortement de la politique, des fonctions de valeur et de récompense ainsi que de la notion de similarité utilisées. Son comportement peut donc être totalement modifié, amélioré en modifiant un seul ou plusieurs de ces facteurs. Par exemple, une politique ne se basant par sur ϵ-*greedy* mais sur *softmax* ou tout autre politique efficace donnerait des résultats différents. Une autre fonction de récompense se basant sur d'autres coefficients ou en combinant ceux utilisés mais différemment ou encore utilisant une vision totalement différente aurait également un impact plus qu'important.

Bibliographie

[1] S. Russel, P. Norvig. *Artificial Intelligence : a Modern Approach (3rd edition)*. Prentice Hall, 2009. ISBN 0-13-604259-7.

[2] Google. Automated cars, September 2010. http://googleblog.blogspot.com/2010/10/what-were-driving-at.html.

[3] AAAI. Dart, December 2011. http://aaai.org/AITopics/Military.

[4] Erkki Oja. Unsupervised learning in neural computation. *Theoretical Computer Science*, 287 :187–207, 2002.

[5] Nizar Grira, Michel Crucianu, Nozha Boujemaa. Unsupervised and semi-supervised clustering : a brief survey. Technical report, INRIA Rocquencourt, 2005. (in A Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (6th Framework Programme)).

[6] J. McQueen. Some methods for classifcation and analysis of multivariate observations. *In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probabilty*, page 281–297, 1967.

[7] L. Kaufman and P. J. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. John Wiley and Sons, 1990. ISBN 0-471-87876-6.

[8] James C. Bezdek, R.Ehrlich and W.Full. Fcm : Fuzzy c-means algorithm. *Computers and Geoscience*, 10 :191–203, 1984.

[9] H. Frigui and R. Krishnapuram. Clustering by competitive agglomeration. *Patter Recognition*, 30(7) :1109–1119, 1997.

[10] P-N Tan, M. Steinbach, V. Kumar. *Introduction to Data Mining*. Addison-Wesley Companion Book Site, 2006.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial data-bases with noise. *In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[12] A.K. Jain and R.C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. ISBN 0-13-022278-X .

[13] A. Hinneburg and D. Keim. An efficient approach to clustering in large multimedia databases with noise. *In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 1998.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1), Year = 1977, Pages = 1-38.

[15] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy*. Freeman, 1973. ISBN 0-7167-0697-0.

[16] B. King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 69 :89–101, 1967.

[17] J.H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58 :236–244, 1963.

[18] O. Chapelle, B. Scholkopf, A. Zien. *Semi-Supervised Learning*. MIT Press, 2006. ISBN 978-0-62-003358-9 .

[19] R.S. Sutton, A.G. Barto. *Reinforcement Learning : An Introduction*. Bradford book, The MIT Press, 1998. ISBN-10 :0-262-19398-1.

[20] I.H. Witten, E. Frank, M.A. Hall. *Data mining : Practical Machine Learning Tools and Techniques*. Elsevier, 2011.

[21] S. Ghosh, A. Nag, D. Biswas, J.P. Singh, S. Biswas, D. Sarkar, P.P. Sarkar. Weather data mining using artificial neural network. *Recent Advances in Intelligent Computational Systems (RAICS)*, pages 192–195, 2011.

[22] April 2012. http://fr.wikipedia.org/wiki/Reconnaissance_de_caract%C3%A8res#Fonctionnement.

[23] W.D. Smart, L.P. Kaelblink. Effective reinforcement learning for mobile robots. *IEEE International Conference on Robotics and Automation*, 4 :3404–3410, 2002.

[24] Sandor Dornbush, Jesse English, Tim Oates, Zary Segall, Anumpam Joshi. Unsupervised and semi-supervised clustering : a brief survey. Technical report, Univeristy of Maryland, Baltimore County, 2007.

[25] J. Layton. How pandora radio works, May 2006. http://computer.howstuffworks.com/internet/basics/pandora.htm.

[26] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *ECOOP'97 — Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0053381.

Annexe A

Ensemble de tests

Auteur	Titre	Genre
Coone	Monstah	Harstyle
Indochine	L'aventurier	Gothic rock
Mylene Farmer	California	Unknown genre
Indochine	Electrastar	Gothic rock
Far East Movement	Like A G6 (Feat. The Cataracs & Dev)	Unknown genre
Bomfunk MC's	B-Boys & Flygirls	Big Beat
John Lennon	Woman (2010 Remaster)	Rock
Bomfunk MC's	Freestyler	Big Beat
Coone	The Way That I Ride	Hardstyle
Arid	If You Go	Pop Rock
Example	Kickstarts	Electonica Pop
Icarus	Main Theme	O.S.T.
Far East Movement	2gether (Feat. Roger Sanchez & Kanobby)	Unknown genre
Adrian Lux	Cant Sleep (Marcus Schossow Presents 1985 Club Remix)	Unknown genre
Ash ft. Avicii	Let Me Show You Love	2012
Avicii	Levels	2012
Basto	Again and Again	2012
Bob Sinclar Feat. Pitbull	Rock The Boat	2012
Carly Rae Jepsen	Call Me Maybe	Unknown genre
Celldweller	Frozen (Celldweller vs Blue Stahli)	Unknown genre
Celldweller	Switchback (Growling Machines Remix)	Unknown genre
Fun.	Carry On	Alternative Rock
Fun.	Some Nights	Alternative Rock
Fun.	We Are Young [feat. Janelle Monáe]	Alternative Rock
Garbage	Cherry Lips (Go Baby Go !)	Unknown genre
Dj Düse	Hub Hub Hubschraubereinsatz	Deutsch
Tim Toupet & Dj Padre	Humba Täterä	Deutsch
Laura Pausini	La Solitudine	Unknown genre
Louise Attaque	Lea	Rock français
Burn it down	Linkin Park	Unknown genre
Lykke Li	I Follow Rivers	Unknown genre
Mickey 3D	Je m'appelle Joseph	Rock français
OG	Friday Night	Techno pop
Otto Knows	Million Voices (Original Mix)	Trance
Seether	Broken ft. Amy Lee	Romantique
Serafin	Day By Day	Alternative Rock
Skillet	Hero	Gothic Rock
Skream ft. Sam Frank	Anticipate	Electronic
Axel Fischer	Traum von Amsterdam (Party Version)	Unknown genre
Unknown artist	we r joung	Unknown genre
Wolfgang Gartner	The Champ (Original Mix)	Unknown genre

Annexe B

Algorithmes supplémentaires

Cette annexe donne les différents algorithmes supplémentaires utilisés par l'agent. Ces algorithmes sont décris en annexe car ils ne sont pas nécessaire à la compréhension du fonctionnement de l'agent. Ils peuvent cependant être intéressants pour les lecteurs désireux d'apprendre les détails du fonctionnement de l'agent.

B.1 Fonction de valeur

Algorithme 4 GetTheMostSimilarSong

Entrée :
- M, la liste des n musiques,
- S, la matrice triangulaire des similarités,
- L, une liste de musiques à éviter (typiquement les musiques déjà lues),
- $song$, une musique

Sortie : la musique la plus similaire à $song$

1: $sim_{max} \leftarrow -1$
2: $i \leftarrow song_{id}$
3: **Pour** toutes les musiques $m \neq song$ dans M **faire**
4: $\quad j \leftarrow m_{id}$
5: $\quad sim_{curr} \leftarrow S_{ij}$
6: $\quad$ **Si** $sim_{curr} > sim_{max}$ et $m \notin L$ **alors**
7: $\quad\quad sim_{max} \leftarrow sim_{curr}$
8: $\quad\quad best \leftarrow m$
9: $\quad$ **fin Si**
10: **fin Pour**
11: **Retourner** $best$

Algorithme 5 Value

Entrée :
- M, la liste des n musiques,
- S, la matrice triangulaire des similarités.
- $song$, une musique
- $depth$, la profondeur

Sortie : $v(song)$ où v est la fonction de valeur

1: $value \leftarrow 0.0$
2: $curr \leftarrow song$
3: $toAvoid \leftarrow$ nouvelle pile
4: $toAvoid.push(curr)$
5: **Pour** i allant de 1 à $depth$ **faire**
6: $best \leftarrow$ GetTheMostSimilarSong(M,S,$toAvoid$,$curr$)
7: $toAvoid.push(best)$
8: $value \leftarrow value f_{\text{REWARD}}(curr, best)$
9: $curr = best$
10: **fin Pour**
11: **Retourner** $\dfrac{value}{depth}$

B.2 Politique

La politique de renforcement s'articule autour de deux algorithmes principaux (les algorithmes 6 et 7) permettant d'obtenir les modifications à appliquer concernant le lien entre deux musiques et concernant l'appréciation d'une musique. Ces modifications sont représentées par une structure **Change** contenant :

- un *type* permettant de définir quelle valeur doit être modifiée (similarité (LINK) ou appréciation (APPRECIATION)),
- une *musique* dans le cas d'une modification d'appréciation, deux dans le cas des similarités,
- un champ représentant le fait qu'il faille augmenter (INCREASE) ou diminuer (DECREASE) la valeur.

Pour faire leur choix, ces algorithmes se basent sur l'humeur et l'action ressentie par le senseur, ces deux valeurs sont représentées par les énumérations **Mood** et **Action**. Voici les différentes valeurs possibles pour ces deux structures :

Action
- PLAY,
- PREVIOUS,
- PAUSE,
- STOP,
- NEXT,
- NEXT_COMPLETE.

Mood
- HAPPY,
- UNHAPPY,
- INDIFFERENT,
- UNKNOWN.

NEXT désigne l'action de l'utilisateur de passer une chanson.
NEXT_COMPLETE désigne la fin d'une chanson.

Algorithme 6 getLinkChangeToDo

Entrée :
- $action$, l'action de l'utilisateur ressentie par le senseur,
- $mood$, l'humeur de l'utilisateur ressentie par le senseur,
- L, la pile des musiques lues par le lecteur telle que la première musique de la pile est la musique en cours de lecture.

Sortie : la modification de similarité à effectuer.

Si L contient au moins deux éléments **alors**
  $current \leftarrow L.pop()$
  $previous \leftarrow L.pop()$
  Si $mood$ = "HAPPY" **alors**
    Retourner Change("INCREASE", "LINK", $previous$,$current$)
  Sinon Si $mood$ = "UNHAPPY" **alors**
    Retourner Change("DECREASE", "LINK", $previous$,$current$)
  Sinon Si $mood$ = "UNKNOWN" **alors**
    Si L contient au moins trois éléments **alors**
      $previous_2 \leftarrow L.pop()$
      Si $action$ = "NEXT" **alors**
        Retourner Change("DECREASE","LINK",$previous_2$,$previous$)
      Sinon Si $action$ = "NEXT_COMPLETE" **alors**
        Retourner Change("INCREASE","LINK",$previous_2$,$previous$)
      fin Si
    fin Si
  fin Si
fin Si
Retourner pas de changement à faire.

(La pile L est considérée comme une copie de la pile réelle du lecteur, les actions *pop()* n'influencent donc pas le comportement du lecteur)

Algorithme 7 getAppreciationChangeToDo

Entrée :
- $action$, l'action de l'utilisateur ressentie par le senseur,
- $current$, la musique vers laquelle le lecteur se dirige,
- $previous$, la musique qui vient d'être terminée/passée.

Sortie : la modification d'appréciation à effectuer.

Si $action$ = "NEXT" **alors**
  Retourner Change("DECREASE", "APPRECIATION", $previous$)
Sinon Si $action$ = "PREVIOUS" **alors**
  Retourner Change("INCREASE", "APPRECIATION", $current$)
Sinon Si $action$ = "NEXT_COMPLETE" **alors**
  Retourner Change("INCREASE", "APPRECIATION", $previous$)
Sinon
  Retourner pas de changement à faire.
fin Si

B.3 Senseur

Le senseur est un *listener* c'est-à-dire qu'il agit quand un événement est généré et qu'il en est notifié. Dans le cas de SmartPlayer, son action est particulièrement importante lorsqu'une musique est terminée/passée. Il maintient en outre deux piles contenant les observations effectuées sur une durée de temps paramétrables (20 actions/humeurs par défaut) : **moods** et **actions**. L'algorithme 8 retrace le comportement du senseur.

Algorithme 8 NextPerformed

Entrée : e, l'événement généré.

Si l'événement a été généré suite à la fin du musique **alors**
  $actions.push$(NEXT_COMPLETE)
Sinon
  $actions.push$(NEXT)
fin Si
Si le **Mood Sensing** est activé **alors**
  $moods.push(senseMood())$
  // $senseMood()$ *fait référence à l'interaction avec l'utilisateur*
Sinon
  $moods.push$(UNKNOWN)
fin Si
Notifier l'agent qu'une nouvelle perception est disponible.

Annexe C

Code source sur CD-ROM