

# LOST VOICE



## — 2023 — RAPPORT DE PROJET

Préparé Par :

**Jeremy MOOTOOVEEREN**

Préparé Par :

**Tang Thanh LONG**

Préparé Par :

**Ismael DRICHE**

Préparé Par :

**Bao Thinh DIEP**

Préparé Par :  
**Sami BELMELLAT**

## Sommaire

### 1. Recherches Et Pistes

- 1.1. L'identification du nombre de locuteurs
- 1.2. Solutions apportées

### 2. Enregistrement Du Son

- 1.1. Bibliotheque Utilisée
- 1.2. Implémentation Dans Le Projet

### 3. Analyse Du Son

- 2.1. Bibliotheque Utilisée
  - 2.1.1. Diarisation/ Diarization
  - 2.1.2. Segment
  - 2.1.3. Cluster
  - 2.1.4. Paramètres acoustique MFCC:
- 2.2. Implémentation Dans Le Projet
  - 2.2.1. Méthode 1 :
  - 2.2.2. Méthode 2 :

### 4. Transcription Vocale

- 3.1. Recherche et Fonctionnement :
- 3.2. Implémentation Dans Le Projet

### 5. Interface Graphique

- 5.1. Les Menus
- 5.2. Labyrinthe
- 5.3. Boite de Dialogue
- 5.4. Animation du personnage

### 6. Diagramme



# 1. Recherches Et Pistes

## 1.1 L'identification du nombre de locuteurs

L'identification du nombre de locuteurs est un problème crucial qui constitue la base de nombreuses applications, allant des assistants vocaux présents dans notre quotidien, tels que Siri et Alexa, aux systèmes de transcription automatique et de contrôle d'accès sécurisé pour protéger des informations sensibles.

## 1.2 Solutions apportées

Au fil du temps, diverses méthodes ont été développées pour résoudre ce problème, et parmi les plus couramment utilisées, on trouve :

- **Méthodes basées sur les Modèles de Mélange Gaussien (GMM)**

Ces méthodes reposent sur deux étapes clés : l'entraînement et la classification. En modélisant les vecteurs de caractéristiques acoustiques extraits d'un enregistrement audio, il est possible de calculer la probabilité qu'ils proviennent d'un nombre donné de locuteurs. Cette approche est largement utilisée en raison de sa simplicité et de son efficacité.

- **Méthodes basées sur les réseaux de neurones**

Ces méthodes, employées par des systèmes tels que Deep Speaker, font appel à des réseaux de neurones pour apprendre les caractéristiques acoustiques de la parole, permettant ainsi l'identification du nombre de locuteurs. Les réseaux de neurones offrent des performances améliorées dans de nombreux cas, en particulier lorsque les données d'entraînement sont suffisantes et représentatives.

- **Méthodes basées sur les Machines à Vecteurs de Support (SVM) :**

Les SVM sont un type d'algorithme d'apprentissage supervisé qui peut être utilisé pour classer les données en différentes catégories. Dans le contexte de l'identification du nombre de locuteurs, les SVM peuvent être formés pour reconnaître les modèles spécifiques aux différents locuteurs, permettant ainsi de distinguer les voix et d'estimer le nombre de locuteurs présents dans un enregistrement audio.



# 1. Recherches Et Pistes

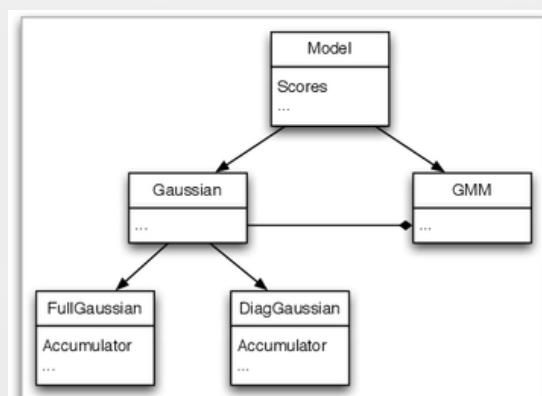
## Quelques méthodes :

- Parmi les alternatives notables, on trouve Kaldi, un kit de développement open-source pour la reconnaissance de la parole, qui propose des outils et des algorithmes avancés pour la diarisation et la classification des voix. Un autre système pertinent est WebRTC Voice Activity Detection (VAD), qui offre des fonctionnalités de détection de la parole en temps réel pour les applications Web.

- Méthodes basées sur les Clusters**

LIUM\_SpkDiarization est un exemple de système qui utilise cette approche. Les méthodes basées sur les clusters découpent l'enregistrement audio en segments, puis les regroupent en clusters en fonction de leur similarité. Cette approche est efficace pour détecter les changements de locuteurs et segmenter l'audio en fonction des différentes voix présentes.

Dans les sections suivantes, nous examinerons plus en détail ces méthodes, en mettant l'accent sur leurs mécanismes, leurs avantages et leurs inconvénients, ainsi que sur les façons dont elles peuvent être adaptées ou combinées pour résoudre le problème d'identification du nombre de locuteurs de manière optimale. Une compréhension approfondie de ces techniques et de leur performance dans diverses conditions est essentielle pour choisir la méthode la mieux adaptée à notre projet spécifique et pour assurer une identification précise et robuste du nombre de locuteurs.





## 2. Enregistrement du son

### 2.1 Bibliothèque utilisée

Nous avons utilisé la bibliothèque javax.Sound qui va nous permettre de capturer le Son via une TargetDataLine et l'enregistrer dans un fichier .wav

### 2.2 Implémentation dans le projet

#### Format Audio :

Afin de capturer notre audio , nous avons utilisé des paramètres bien spécifiques pour la meilleur efficacité possible :

- Fréquence Audio : 16Khz
- BitRate de l'audio : 16 (Le nombre de bits par seconde dans le fichier audio)
- Audio en Mono channel
- Signature de l'audio : oui
- La data est stocké dans le bit le plus grand : oui

#### Enregistrement du son :

Cela se déroule dans la class Recorder , là où on a la fonction void record() qui permet de capturer le son . Tout d'abord on récupère le format audio décrit ci-dessus Par la suite, on vérifie si notre ligne de capture est bel et bien supportée par le système de notre utilisateur .

On ouvre une nouvelle ligne avec le format audio , et on commence la capture du son . En parallèle on récupère le flux de bits capturés par la ligne dans un AudioInputStream , qu'on va écrire dans un fichier de type Wav au fur et à mesure de la capture , et tant que la ligne n'a pas été fermée .

On lance ce processus dans un Thread en parallèle du Thread principale pour éviter que le système se concentre sur l'unique tâche d'enregistrer du son et néglige toutes les tâches restantes derrière , ce Thread s'arrête quand la fonction void stopRecording() est appellé qui va arrêter la capture du son , et fermer la ligne de capture.



## 3. Analyse Du Son

C'est la partie principale de notre projet et la plus importante , son but est l'identification du genre et du nombre de locuteurs dans un enregistrement vocal donné.

### 3.1 Bibliothèque Utilisée

Afin de permettre l'analyse du son et déterminer le genre et nombre de locuteurs d'un enregistrement audio , nous avons utilisé " Lium SpkDiarization " qui est une librairie spécialisée dans la **diarization** , **traitement du signal audio** ainsi que la **reconnaissance vocale** et de **genre** via des **paramètres acoustiques MFCC** bien définis .

Écrite en java , elle a été développée par les chercheurs de **l'université Dumont** . Voici quelques notions très importantes à savoir :

#### **MFCC (Mel-Frequency Cepstral Coefficients)**

Les MFCC permettent d'extraire des caractéristiques sonores qui reflètent la perception humaine des fréquences. Le processus pour obtenir les MFCC comprend plusieurs étapes, notamment la division du signal en trames, l'application de la transformée de Fourier à court terme, l'utilisation de l'échelle de fréquence de Mel et la conservation des premiers coefficients cepstraux.

#### **Diarisation/ Diarization**

La diarisation est un processus en traitement automatique de la parole qui consiste à segmenter et à étiqueter un enregistrement audio en fonction des différents locuteurs présents. Cette procédure comprend généralement plusieurs étapes, dont :

- Segmentation
- Extraction de caractéristiques
- Regroupement (clustering)

#### **Segment:**

Dans le contexte de LIUM, un segment fait référence à une portion d'un enregistrement audio qui a été isolée et étiquetée en fonction de certaines caractéristiques, comme la présence d'un locuteur spécifique. LIUM utilise des algorithmes pour découper et regrouper les segments audio en fonction des locuteurs. Chaque segment est identifié par un point de départ, une durée et un identifiant de locuteur (Homme ou Femme).



## 3. Analyse Du Son

### Cluster

Un cluster est utilisé pour représenter un locuteur unique dans un enregistrement audio. Les clusters sont utilisés pour regrouper les segments audio similaires et sont ensuite analysés pour extraire des informations telles que le genre du locuteur. Les clusters sont gérés à l'aide de la classe ClusterSet et de la classe Cluster du package.

### Paramètres acoustique MFCC:

Argument des paramètres : **audio16Khz2sphinx:sphinx,1:1:0:0:0:0,18,0:0:0:0 :**

Les paramètres s, e, d, de, dd, dde correspondent respectivement aux valeurs statiques, à l'énergie, aux deltas, à l'énergie delta, aux delta-deltas et à l'énergie delta-delta. Les valeurs [ 0 , 1 , 2 , 3 ] décrivent l'état de chaque élément (présent ou non, chargé à partir du disque ou supprimé après le chargement).

Voici les paramètres les plus importants :

- **audio16Khz2sphinx:sphinx :**

Afin d'effectuer la diarization selon les paramètres acoustiques fournis par Lium nous fournit les outils internes qui sont présents dans sphinx.

- **Energy (Énergie) :**

L'énergie d'un signal audio est une mesure de l'amplitude du signal sur une certaine période de temps. Dans le contexte des caractéristiques audio, elle est généralement incluse pour représenter l'intensité globale du signal audio.

- **Delta :**

Les coefficients delta ( $\Delta$ ) représentent la différence entre les coefficients de caractéristiques adjacents dans le temps. Ils sont utilisés pour capturer l'information sur la dynamique temporelle du signal audio, comme la vitesse à laquelle les caractéristiques changent. Les coefficients delta sont souvent utilisés en combinaison avec les caractéristiques statiques pour améliorer la performance des modèles de reconnaissance vocale et de diarization.



## 3. Analyse Du Son

- **Delta-delta :**

Les coefficients delta-delta ( $\Delta\Delta$ ) représentent la différence entre les coefficients delta adjacents dans le temps. Ils sont utilisés pour capturer l'information sur l'accélération des changements dans les caractéristiques audio et peuvent aider à améliorer la performance des modèles de reconnaissance vocale et de diarization.

- **Delta-delta energy (Énergie delta-delta) :**

L'énergie delta-delta est la différence entre les valeurs d'énergie delta adjacentes dans le temps. Elle représente la variation de la variation de l'énergie du signal audio au fil du temps et peut être utilisée pour capturer des informations sur l'accélération des changements d'énergie.

### 3.2 Implémentation dans le projet

L'analyse vocale est réalisée dans le fichier AudioAnalyser.java. Ce dernier illustre la procédure que nous avons suivie pour développer deux méthodes dédiées au traitement de la voix, dans le but précis de déterminer le nombre de locuteurs masculins et féminins dans un enregistrement vocal donné. Ces deux méthodes sont concrétisées par les deux fonctions "int[] analysis1(String path)" et "int[] analysis2(String path)" dans AudioAnalyser.

Ces fonctions ont été conçues pour analyser les fichiers audio, extraire les caractéristiques acoustiques pertinentes, et appliquer le processus de diarisation pour identifier le nombre et le genre des locuteurs. Elles sont destinées à être utilisées de manière interchangeable, offrant une certaine flexibilité en fonction des exigences spécifiques de l'analyse audio.

En somme, l'approche adoptée pour l'implémentation de notre système s'appuie fortement sur la bibliothèque LIUM SpkDiarization et son application pour analyser les caractéristiques acoustiques. Les résultats obtenus à partir de ces analyses sont ensuite utilisés pour guider le mouvement d'un personnage à travers un labyrinthe, en fonction du nombre et du genre des locuteurs identifiés. Cette approche démontre non seulement l'applicabilité pratique de la diarisation dans un environnement de jeu, mais offre également une illustration pertinente des potentialités offertes par l'analyse sonore dans un contexte plus large.



# 3. Analyse Du Son

Nous avons pu élaboré deux méthodes d'analyse

## 1. Méthode 1 :

Cette première méthode s'appuie sur des fonctions déjà existantes dans **LIUM Spk Diarization-8.4.1.jar**. Cette méthode va se dérouler en deux parties :

- Analyse Audio par lium
  - Interprétation des résultats de l'analyse du fichier .seg

Avant d'y procéder , on a d'abord créé une classe ‘**Terminal**’ qui va nous permettre d'exécuter des commandes shell .

Par la suite on exécute la commande suivante :

```
“/usr/bin/java -Xmx2048m -jar ./LIUM_SpkDiarization-8.4.1.jar  
finputDesc=audio16Khz2sphinx:sphinx,1:1:0:0:0:0,18,0:0:0:0  
--fInputMask=./src/main/java/com/VocalMaze/Records/Audio.wav  
--fOutputMask=./src/main/java/com/VocalMaze/Analysis/Analysis/seg  
--doCEClustering Audio” grâce à la classe citée ci-dessus .
```

- **--fInputDesc** : Cet argument sera les paramètres acoustiques ‘MFCC’ que notre analyse va devoir utiliser , ces derniers qui sont montrés dans cette commande ont été trouvés après une série de tests sur différents enregistrements vocaux.
  - **--fInputMask** : Cet argument sera le chemin relatif ou absolu vers notre fichier audio sous format Wav
  - **--fOutputMask** : Cet argument sera le chemin relatif ou absolu de l'endroit où le fichier résultant de l'analyse de notre audio sera mis, ce dernier est un fichier texte .seg comportant les résultats de l'analyse qu'il faudra interpréter par la suite
  - **--doCEClustering** : Cet option permet de regrouper en Cluster tous les segments qui sont similaires . ainsi minimiser le risque d'erreur de la diarisation

Une fois la commande exécutée , on devra interpréter le fichier .seg qui représente les résultats de la diarization et de l'analyse vocale , dont la structure est présentée comme suit:  
Plusieurs lignes représentent des segments de Clusters , on retrouve ‘;;’ au début de chaque segment d'un nouveau Cluster



### 3. Analyse Du Son

Chaque segment est représenté comme suit :

cluster:S0 Audio 1 7 364 F S U S0

1. Colonne 1 : Le nom du cluster
2. Colonne 2 : Le nom de l'audio analysé
3. Colonne 3 : Le début du segment (centième de secondes)
4. Colonne 4 : La duré du segment (centième de secondes)
5. Colonne 5 : Le genre du cluster(M ou F)
6. Colonne 6 : Le type de la bande (Studio ou Téléphone)
7. Colonne 7 : Le type de l'environnement du son
8. Colonne 8 : Le label du Cluster

Ainsi il est possible de déterminer le nombre exact de locuteurs dans un enregistrement donné , ainsi que le début et fin de chacun de leurs temps de paroles détecté par la Diarization de Lium. La classe SegAnalyser possède une fonction “ int [ ] analysis(String path) ” qui prend en paramètre un chemin vers un fichier .seg existant et retourne un tableau de taille 2 contenant à la case 0 le nombre de locuteurs Homme et à la case 1 le nombre de locuteurs Femme!

#### 2. Méthode 2 :

Cette méthode est un traitement de Lium manuel , en reprenant des fonctions dans **LIUM\_SpkDiarization-8.4.1.jar**.

**executeForResult(String audioFile) throws Exception :**

La méthode executeForResult(String audioFile) prend en entrée un chemin de fichier audio sous forme de chaîne de caractères et effectue plusieurs vérifications avant de procéder à l'analyse du fichier.

- Prend en entrée un chemin de fichier audio sous forme de chaîne de caractères.
- Vérifie si le fichier existe. Si ce n'est pas le cas, une exception est levée.
- Vérifie si le format du fichier audio est WAV. Si ce n'est pas le cas, une exception est levée.
- Appelle la méthode executeForResult avec un tableau de chaînes contenant l'argument -fInputMask= suivi du chemin du fichier audio.
- Si toutes les conditions sont remplies, elle appelle la méthode executeForResult avec un tableau de chaînes contenant l'argument --fInputMask= suivi du chemin du fichier audio.



### 3. Analyse Du Son

**resultFromSegment(String segmentFile)** : analyse les fichiers de segmentation et renvoie un objet Result contenant les nombres de voix détectées pour chaque genre.

- Prend en entrée un chemin de fichier de segmentation.
- Vérifie si le fichier existe. Si ce n'est pas le cas, une exception est levée.
- Crée un nouvel objet Result.
- Lit le fichier de segmentation ligne par ligne.
- Pour chaque ligne contenant un cluster et un score, incrémente le compteur de genre approprié dans l'objet Result.
- Renvoie l'objet Result avec les nombres de voix détectées pour chaque genre.

**executeForResult(final String[] args) throws Exception :**

- Prend en entrée un tableau de chaînes de caractères contenant les arguments de la commande.
- Crée un nouvel objet Diarization et un nouvel objet Result.
- Initialise les paramètres de la bibliothèque LIUM SpkDiarization avec les arguments fournis.
- Effectue plusieurs étapes pour analyser le fichier audio :
- Initialise le système de diarization.
- Sépare les hypothèses en utilisant MainTools.splitHypothesis.
- Pour chaque hypothèse (représentée par un ClusterSet) :
- Charge les fonctionnalités audios.
- Effectue un contrôle de cohérence.
- Réalise la segmentation.
- Effectue un clustering linéaire.
- Effectue un clustering hiérarchique.
- Décode les résultats.
- Sépare les segments de parole.
- Déetecte le genre des voix.
- Incrémente les compteurs de genre dans l'objet Result pour chaque voix détectée.
- Renvoie l'objet Result avec les nombres de voix détectées pour chaque genre.

Cette méthode est le cœur de l'analyse vocale. Elle effectue plusieurs étapes pour analyser le fichier audio et détecter le genre des voix présentes. Les étapes incluent l'initialisation, la segmentation, le clustering et la détection du genre. Le résultat final est un objet Result contenant le nombre de voix détectées pour chaque genre.

La classe Result est une classe interne qui stocke le nombre de voix détectées pour chaque genre (inconnu, masculin et féminin). Elle contient une méthode increase(String gender) pour incrémenter le nombre de voix détectées pour un genre spécifique et une méthode toString() pour renvoyer une représentation sous forme de chaîne de l'objet Result.



## 4. Transcription Vocale

### 4.1. Recherche et Fonctionnement

#### 1. Amazon Transcribe :

Notre objectif était de pouvoir traduire les mots d'un audio donnée pour les récupérer dans un String que nous utiliserons pour détecter les directions énoncées par les joueurs. En outre, après de multiples recherches sur la transcription et sa capacité à être utilisé dans nos programmes Java, nous avions porté notre regard sur une bibliothèque nommée "Alizée". Seulement, celle-ci nécessite l'utilisation de modèles audio pour mettre d'associer une séquence audio à un mot.

Voyant que cette méthode était assez lourde puisque dans notre utilisation, nous voulions une réelle transcription des audios en String et non pas de mot clés, ce qui aurait nécessité une grosse quantité de modèles, nous avons réfléchi à une autre méthode.

C'est ainsi que pour parer à ce problème, nous avons songé à utiliser les services AWS et plus précisément la méthode de transcription d'Amazon, notamment utilisée par Alexa.

La bibliothèque que nous utilisons se nomme : "**Amazon Transcribe**"

#### 2. Intérêt et Fonctionnement

Amazon Transcribe est un service cloud de reconnaissance vocale automatique fourni par Amazon Web Services (AWS) qui permet de convertir facilement des fichiers audio en texte. Une bibliothèque qui utilise des algorithmes avancés d'intelligence artificielle pour transcrire avec précision et rapidité l'audio enregistré dans des fichiers audio en texte.

Pour transcrire le fichier audio, il faut lui fournir le fichier audio par différentes méthodes comme l'upload par interface web ou par API comme nous procédons. Vous pouvez choisir parmi une variété de formats audio pris en charge, notamment MP3, WAV, FLAC et OGG. Une fois que le fichier audio est uploadé, Amazon Transcribe utilise des algorithmes avancés de reconnaissance vocale pour transcrire le contenu audio en texte. Le service peut reconnaître et reconnaître plusieurs locuteurs même dans les situations les plus complexes et retranscrire les paroles prononcées en temps réel.

Le service peut également tenir compte des accents, du bruit de fond et des erreurs de prononciation pour garantir une précision de transcription maximale.

Une fois la transcription terminée, le texte sera disponible au format JSON qui peut être parsé pour en exploiter les phrases.

En résumé, Amazon Transcribe est un outil efficace et facile à utiliser pour transcrire des fichiers audio.



## 4. Transcription Vocale

### 4.2. Implémentation dans le projet

#### 1. Fonction transcriber()

La fonction transcriber fonctionne de la manière suivante :

Tout d'abord, le programme configure les informations d'identification AWS (clé d'accès et clé secrète) pour accéder aux services AWS.

Ensuite, il crée un client AmazonTranscribe pour effectuer la transcription de la parole en texte et un client AmazonS3 pour télécharger le fichier audio dans le cloud. Le programme générera un nom de tâche de transcription aléatoire et définira la langue de transcription sur le français.

En récupérant le fichier audio que l'on aura au préalable enregistré par l'action du joueur, on initialise les métadonnées de l'audio au format MP3 (on convertit plus tard le WAV en MP3) car l'ont doit fournir aux serveurs AWS la nature du fichier sinon il l'utilisera en tant que fichier binaire.

Par la suite, on upload les fichiers audio vers un bucket S3 dont le nom est également généré de manière aléatoire. Ce programme utilise la classe WavToMp3Converter pour convertir les fichiers audio du format WAV au format MP3 afin de réduire le délai de traduction.

On crée une demande pour démarrer une tâche de transcription à l'aide du nom de la tâche de transcription, de la langue de transcription, du format de fichier audio et de l'URI du fichier audio stockés dans Amazon S3. Le programme vérifie périodiquement l'état de la transcription en fonction du nom de la tâche de transcription jusqu'à ce que l'état soit "COMPLETED".

Une fois la transcription terminée, le programme utilisera le nom de la tâche de transcription pour obtenir les résultats de la transcription et les enregistrer dans un fichier texte. Renvoie le résultat de la transcription du texte sous la forme d'une chaîne après un parsage du fichier JSON

En résumé, ce programme utilise les services AWS pour effectuer une transcription voix-texte en plusieurs étapes, y compris la configuration des informations d'identification AWS, le transfert de fichiers audio, le démarrage d'une tâche de transcription et la récupération des résultats de la transcription.



## 4. Transcription Vocale

### 2.Fonction getDirection()

Après avoir utilisé la fonction transcriber(), celle-ci renvoie un String qui contient la traduction phrase par phrase de l'audio donnée en argument. Cette fonction permet de parser le String et récupérer les instructions "HAUT", "BAS", "GAUCHE" ou "DROITE" pour les intégrer dans une LinkedList en utilisant l'enum que nous avons réalisé pour les commandes.

Voici comment fonctionne la fonction :

La méthode commence par diviser la chaîne de caractères en un tableau de mots à l'aide de l'expression régulière "\s+".

Ensuite, elle parcourt chaque mot du tableau words et vérifie s'il contient les chaînes de caractères "haut", "bas", "gauche" ou "droite". Si un mot contient l'une de ces chaînes, la méthode ajoute la direction correspondante dans un objet ArrayList<Direction>.

Finalement, la méthode convertit l'objet ArrayList<Direction> en un tableau de direction et le renvoie.

A noter que l'ordre des directions énoncées par les joueurs demeure le même étant donné que le parsing s'effectue du début de la phrase vers la fin, ainsi nous n'avons aucun souci quant à l'ordre d'exécution des commandes.

Ce programme permet donc de récupérer les commandes après la transcription.

### 3.Conclusion

En résumé, nous utilisons la bibliothèque "Amazon Transcribe" afin de transcrire le fichier audio que les joueurs enregistrent pour en récupérer un String qui est la traduction de l'audio en texte puis nous procédons à un parsing qui permet de récupérer les commandes énoncées par les joueurs. Ces commandes serviront à faire bouger le personnage dans les directions se situant dans la LinkedList de Commandes.

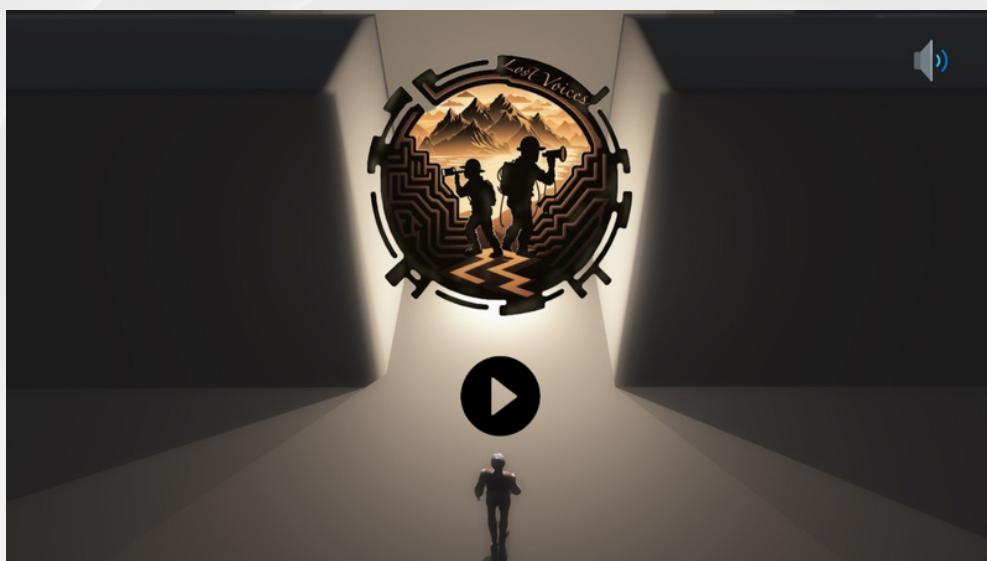


## 5. Interface Graphique

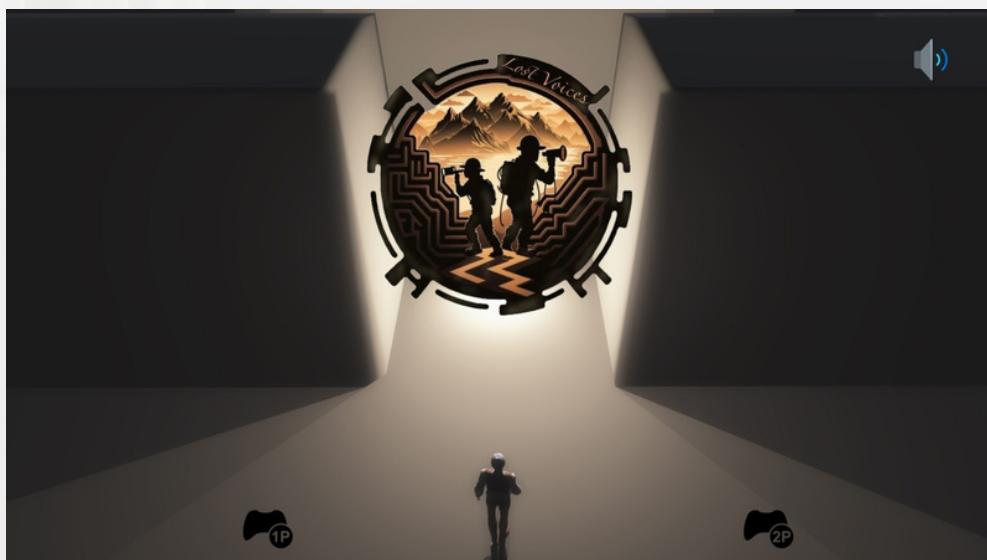
### 5.1. Les Menus

Nous avons réalisé des menus en java SWING/AWT afin de permettre une navigation simple dans le jeu pour l'utilisateur , tout en leur offrant une bande son créée par nous mêmes avec des thèmes et ambiances pour renforcer l'immersion .

En voici quelques extraits :



**Menu d'accueil sur lequel on peut mute la musique et lancer le jeu**



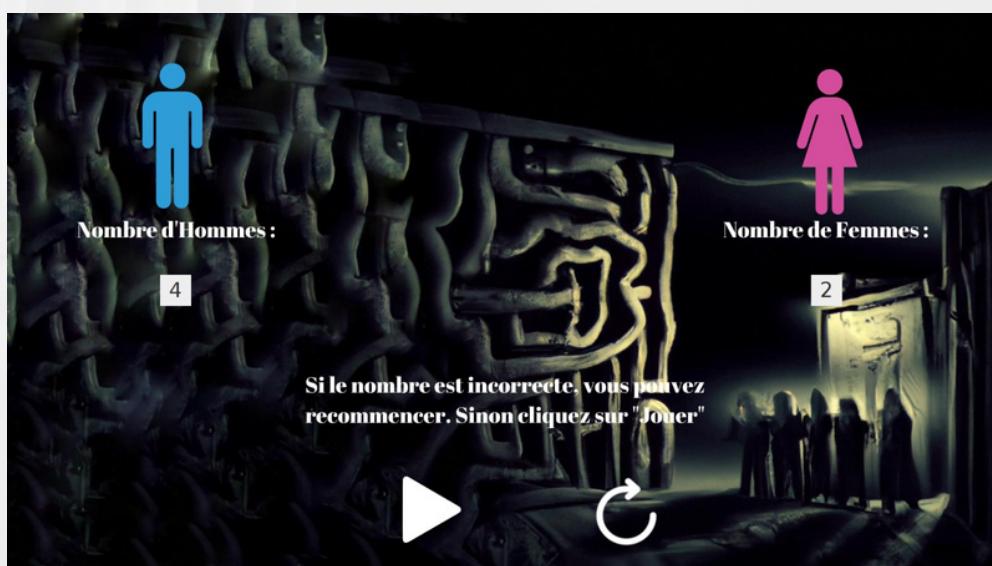
**2ème Menu sur lequel on choisit entre un jeu en solo ou en équipe**



## 5. Interface Graphique



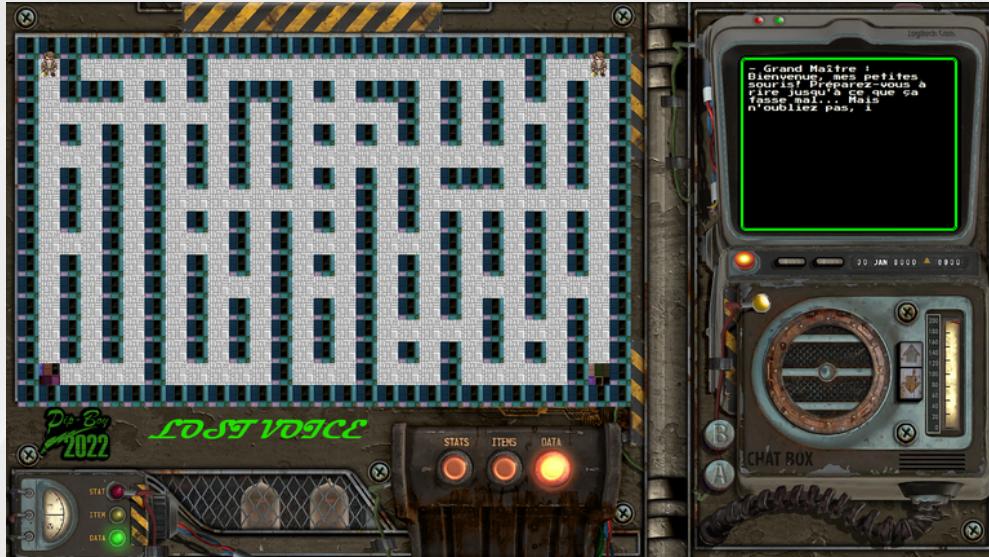
**3ème Menu qui permet d'initialiser l'analyse de Lium afin de l'entraîner mais aussi de déterminer le nombre de femmes et d'hommes**



**4ème Menu qui permet de vérifier le nombre d'hommes et de femmes**



## 5. Interface Graphique



Interface du jeu sur lequel on joue

### 5.2 Labyrinthe

Notre jeu étant basé sur la reconnaissance vocale et le déplacement sur un chemin, on a donc adapté cela à un labyrinthe. Pour se faire, soit il est idée de créer plusieurs chemins à la main et donc un nombre limité de labyrinthe, soit trouver une façon de créer des chemins aléatoires et donc une infinité de labyrinthes différents.

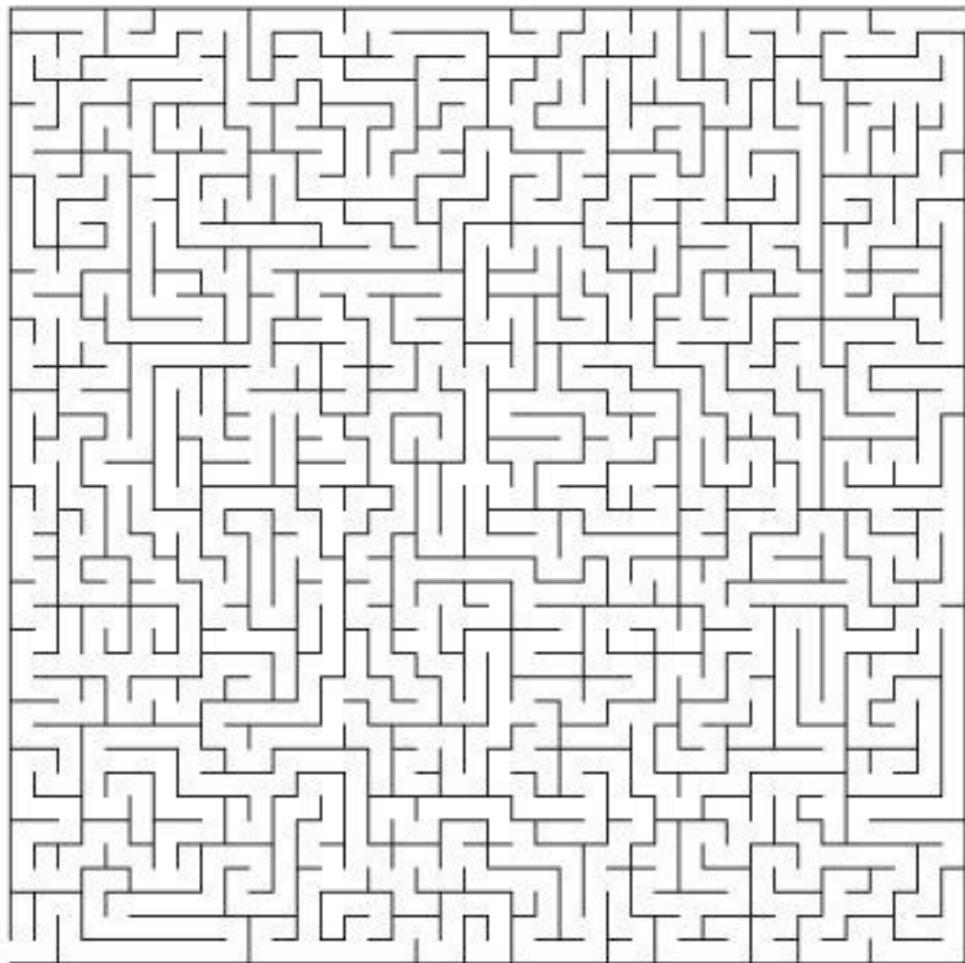
En optant pour le deuxième choix, on trouve alors plusieurs noms d'algorithme notamment, l'algorithme de Prim.

C'est en s'inspirant de cet algorithme, qui nous a permis de créer des fonctions permettant de générer des labyrinthe des aléatoires et cela nous prouve que cet algorithme est très satisfaisant pour générer des labyrinthes.

On utilise un tableau à 2 dimensions afin de représenter le labyrinthe et faciliter l'affichage graphique de celui-ci, et les différentes fonctions permettant au jeu de fonctionner tel que le déplacement, les murs, etc.



## 5. Interface Graphique



**Voici le type de labyrinthe que l'on a implémenté. Un labyrinthe comme on peut le trouver dans certains jeux ou magazine. En outre, l'idée ici est juste de reproduire le principe du labyrinthe mais en axant le jeu sur l'aspect directionnel**



## 5. Interface Graphique

### 5.3 Boîte de Dialogue

Boîte de dialogue ou PopUp est un menu de dialogue conçu pour permettre aux joueurs d'interagir davantage avec le jeu. Le menu PopUp a pour objectif d'améliorer l'expérience de jeu en offrant aux joueurs une interface interactive et dynamique. Il permet d'afficher des dialogues, des instructions et des messages contextuels tout au long du jeu. Grâce à ce menu, les joueurs peuvent mieux comprendre les objectifs du jeu, les tâches à accomplir et les défis à relever.

Dans ce jeu en particulier, le menu PopUp est utilisé pour présenter les échanges entre les personnages et le Grand Maître, un antagoniste qui guide les joueurs à travers le labyrinthe, et les défie de trouver la sortie. Les dialogues sont conçus pour ajouter une dimension narrative et émotionnelle au jeu, tout en fournissant des indices et des conseils pour aider les joueurs.

En outre, le menu PopUp est intégré de manière fluide dans l'interface utilisateur, en étant placé à l'est de l'écran de jeu. Il est conçu pour être facile à lire et à naviguer, sans perturber l'expérience de jeu globale.

Dans l'ensemble, en offrant des interactions supplémentaires et un dialogue captivant, il contribue à créer une expérience de jeu plus riche et plus intéressante pour les utilisateurs.

- Grand master : Salutations, mes chers aventuriers! Préparez-vous à trembler de terreur. Maintenant, commençons le jeu. Vous devez trouver la sortie avant que je ne vous trouve. Ahahaha... Vous êtes à moi maintenant. Osez-vous relever le défi ? Hahahaha!

- Jeu : Dans un premier temps, vous devez parler chacun votre tour afin de vous reconnaître, ainsi gagner du temps de parole. Appuyez sur R pour commencer l'enreg

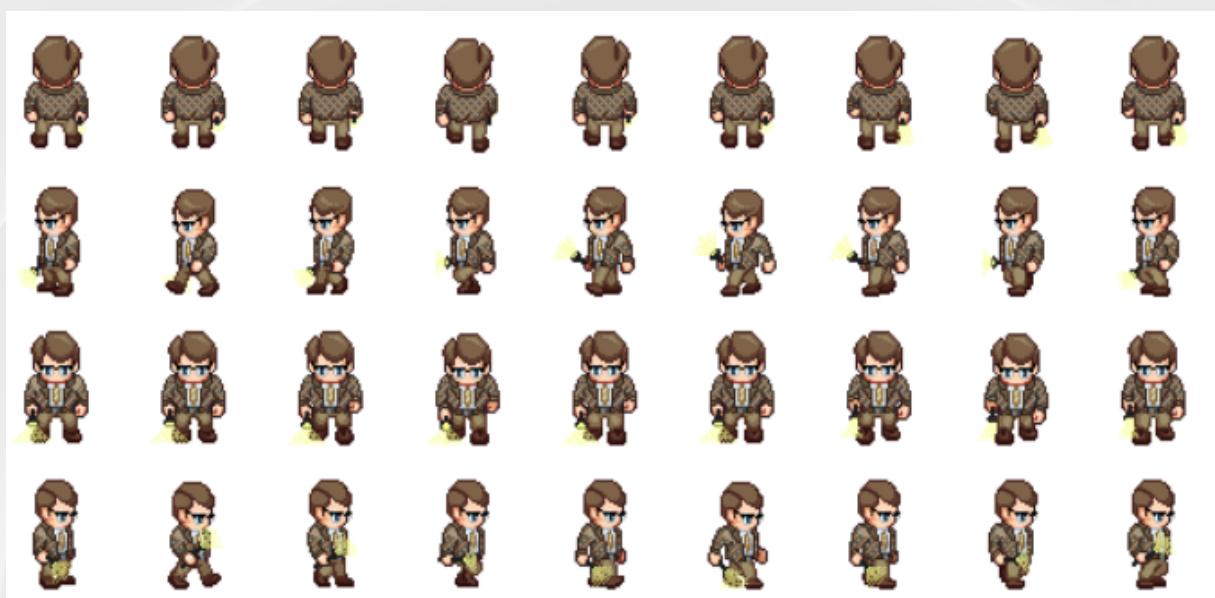
PopUp qui apparaît sur le bord droit du jeu qui permet de donner des directives mais aussi de pouvoir donner des informations sur le jeu



## 5. Interface Graphique

### 5.4 Animation du Personnage

Pour l'animation, nous avons déjà commencé par choisir un personnage :



On commence par lire cette image dans le constructeur de la classe `labyrintheView`. Ensuite, avec la méthode `decoupeImage()` nous allons stocker dans un tableau à deux dimensions de `bufferedImage` chaque image correspondant à un mouvement du personnage. Chacune des images de notre tableau est de dimension 64x64. Pour connaître les dimensions de chaque image on fait largeur / 9 (il y a neuf mouvements) et hauteur / 4 (il y a quatre directions HAUT, BAS, GAUCHE et DROITE).

Ensuite, pour jouer l'animation à l'écran nous appelons la méthode `movePlayer(Direction dir, int steps)`. Cette méthode va lire en boucle très rapidement le double tableau de `BufferedImage` et les afficher à l'écran avec la méthode `drawImage(Image img, int x, int y, ImageObserver observer)` en fonction de la direction et du nombre de pas.



## 6. Diagramme

