



# **RAPPORT PROJET TUX**

**KHALIL IBRAHIM GOUKOUNI**

**ABDRAMAN ABAKAR**

## Introduction

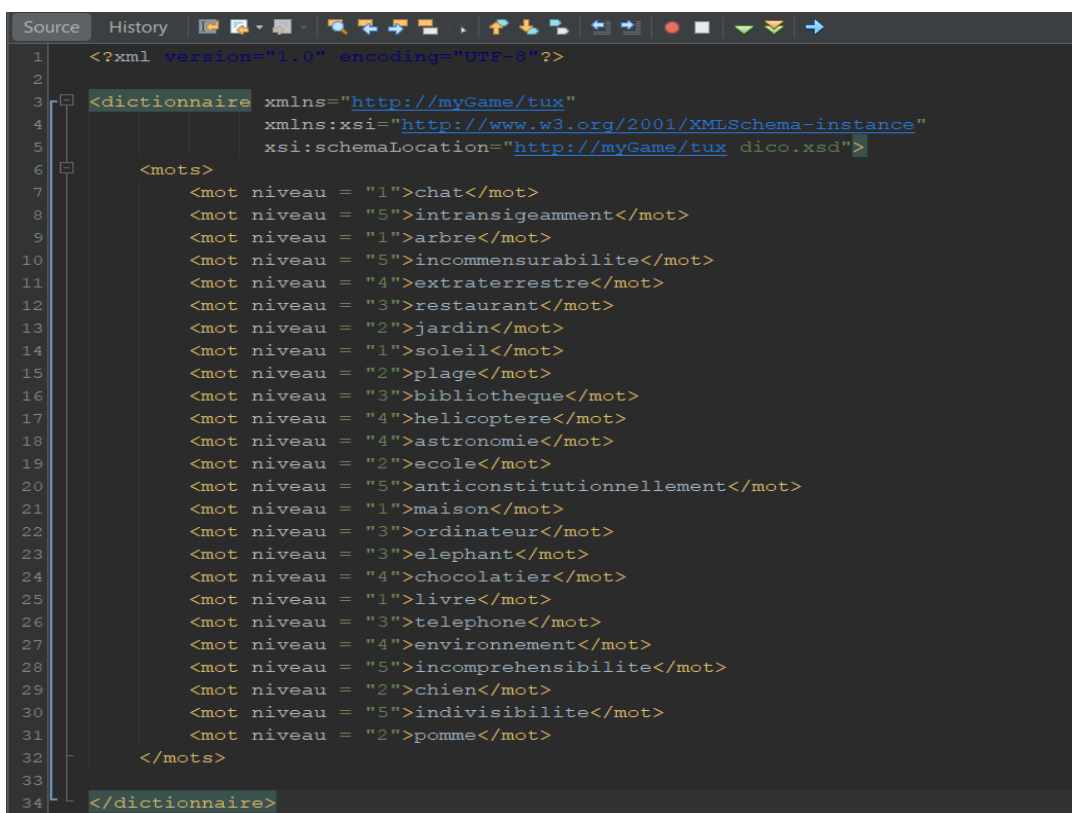
Le projet **Tux LetterGame** est un mini-jeu 3D permettant l'apprentissage ludique de l'orthographe en cherchant des lettres d'un mot avec un personnage, tout cela dans un temps imparti.

### I- Quelques notions en xml

Pendant le développement de ce projet, nous avons appliqué nos connaissances en XML, XSD et XSLT acquises au cours de divers travaux pratiques. Nous avons formalisé plusieurs fichiers, tels que **dico.xml**, **dico.xsd**, **dico.xslt**, **profil.xml**, **profil.xsd**, **profil.xslt** et **plateau.xml**.

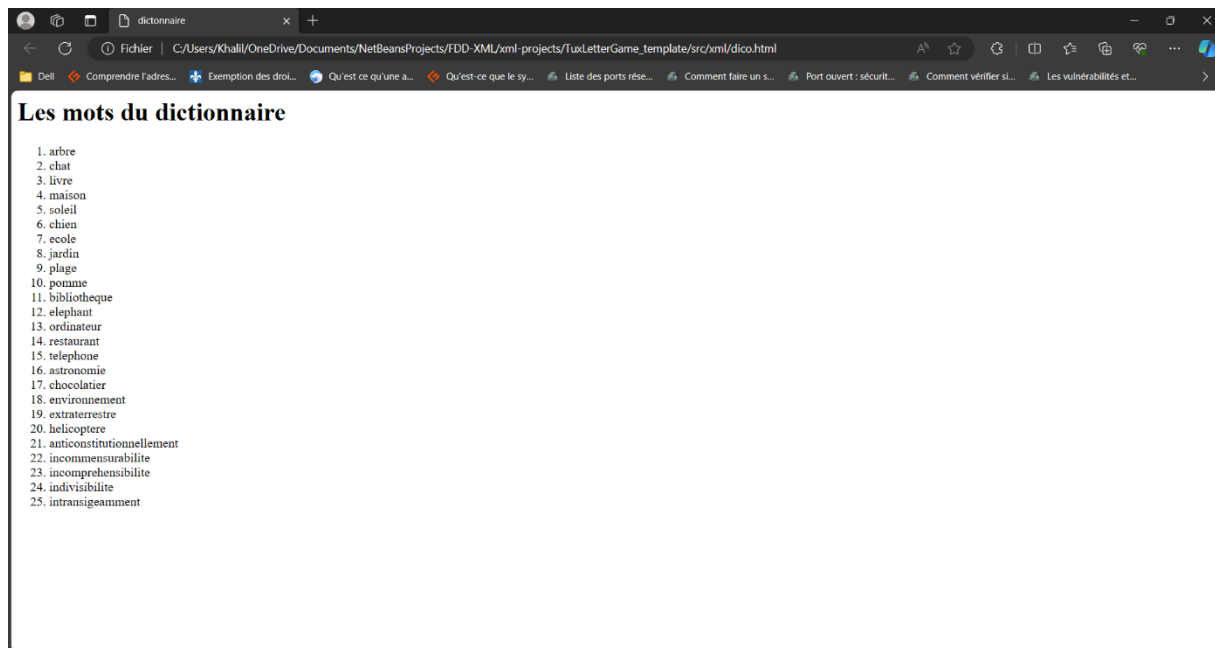
En ce qui concerne les fichiers du dictionnaire (dico) : le fichier **dico.xsd** définit les types (**complexes et simples**) des éléments présents dans **dico.xml**. Cette approche a permis de créer un fichier **dico.xml** bien structuré. Le fichier **dico.xsl**, quant à lui, transforme le fichier en une page **HTML** en triant les mots de **dico.xml** par niveau, et pour chaque niveau, par ordre alphabétique. Cette transformation rend la consultation du dictionnaire plus conviviale en présentant les informations de manière organisée et facilement accessible.

Notre fichier **dico.xml** :



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <dictonnaire xmlns="http://myGame/tux"
4             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5             xsi:schemaLocation="http://myGame/tux dico.xsd">
6   <mots>
7     <mot niveau = "1">chat</mot>
8     <mot niveau = "5">intransigeamment</mot>
9     <mot niveau = "1">arbre</mot>
10    <mot niveau = "5">incommensurabilite</mot>
11    <mot niveau = "4">extraterrestre</mot>
12    <mot niveau = "3">restaurant</mot>
13    <mot niveau = "2">jardin</mot>
14    <mot niveau = "1">soleil</mot>
15    <mot niveau = "2">plage</mot>
16    <mot niveau = "3">bibliotheque</mot>
17    <mot niveau = "4">helicoptere</mot>
18    <mot niveau = "4">astronomie</mot>
19    <mot niveau = "2">ecole</mot>
20    <mot niveau = "5">anticonstitutionnellement</mot>
21    <mot niveau = "1">maison</mot>
22    <mot niveau = "3">ordinateur</mot>
23    <mot niveau = "3">elephant</mot>
24    <mot niveau = "4">chocolatier</mot>
25    <mot niveau = "1">livre</mot>
26    <mot niveau = "3">telephone</mot>
27    <mot niveau = "4">environnement</mot>
28    <mot niveau = "5">incomprehensibilite</mot>
29    <mot niveau = "2">chien</mot>
30    <mot niveau = "5">indivisibilite</mot>
31    <mot niveau = "2">pomme</mot>
32  </mots>
33 </dictonnaire>
```

Transformation xsl du dico.xml en une page HTML :



Concernant les fichiers liés aux profils des joueurs : le fichier **profil.xsd** établit les types (**complexes et simples**) des éléments inclus dans **profil.xml**. Cette méthodologie a conduit à la création d'un fichier **profil.xml** bien structuré. En complément, le fichier **profil.xsl** réalise une transformation du fichier en une **page HTML**, présentant les parties jouées par un joueur sous la forme d'un tableau. Cette représentation tabulaire facilite la visualisation et l'analyse des activités de jeu d'un joueur donné.

Transformation xsl du **profil.xml** en une **page HTML** :

The screenshot shows a web browser window with the address bar displaying the file path: C:/Users/Khalil/OneDrive/Documents/NetBeansProjects/FDD-XML/xml-projects/TuxLetterGame\_template/src/xml/profil.html. The page content is titled 'Alban' and 'Parties jouées'. It displays a table with the following data:

Date	Mot	Niveau	Score
2013-12-10	cle	1	5
2013-12-11	indice	2	0
2013-12-04	loup	2	10

Le fichier **plateau.xml** contient les informations nécessaires à la mise en place de l'environnement du jeu Tux LetterGame.

## II- Objectif du Tux LetterGame

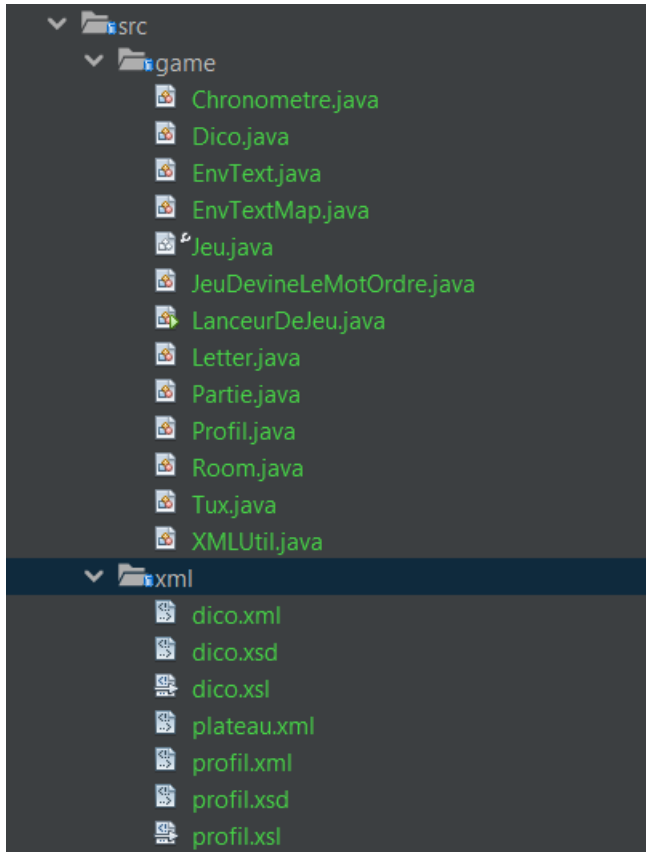
Le déroulement du jeu prendra la forme suivante :

- 1 – Le jeu propose un premier menu permettant :
  - 1- De charger le profil d'un joueur existant
  - 2- De créer un nouveau joueur
  - 3- De quitter le jeu
- 2 – Une fois le joueur actif, le jeu propose un deuxième menu permettant :
  - 1- De lancer une nouvelle partie
  - 2- De charger une partie existante
  - 3- D'ajouter une nouvelle partie
  - 4- De sortir de la partie courante
  - 5- De quitter le jeu
- 3 – Si une nouvelle partie est demandée, le jeu permet de sélectionner un niveau de difficulté. Ce niveau concernera principalement la difficulté du mot à trouver.
- 4 – Un mot est alors chargé depuis le dictionnaire. Dans notre cas, dans la classe Jeu permet de lire le fichier dico.xml pour pouvoir sélectionner les mots selon le niveau entré par le joueur.
- 5 – Une instruction Trouver ce mot accompagner **du mot à trouver** sont montés pendant 5 secondes puis une partie est lancée
- 6 – Un personnage et des lettres apparaissent. Les lettres sont posées au hasard dans l'environnement. Le joueur peut déplacer son personnage avec les flèches de direction ou en utilisant le clavier.
- 7 – Le jeu s'arrête :
  - Soit parce que le temps imparti est fini
  - Soit parce que les lettres ont toutes été choisies

Dans les deux cas, le jeu enregistre la partie et revient au 2<sup>ème</sup> menu.

### III- Architecture

Le projet **TuxLetterGame** est organisé de manière modulaire, favorisant une structure claire et compréhensible. Les principaux composants du projet sont répartis dans des répertoires distincts, chacun remplissant un rôle spécifique dans la logique du jeu. Voici donc une capture d'écran contenant les différents répertoires qui composent notre jeu :



Le répertoire **src** qui contient le **package game** et le **répertoire xml** ;

#### 1 - Le package **game** :

- a- **Le package "game"** constitue le cœur du projet Tux LetterGame, regroupant les principales classes et composants responsables de la logique du jeu. Les différentes classes dans ce package interagissent étroitement pour créer une expérience de jeu fluide. Voici une description détaillée de certaines des classes présentes dans le package "game" :
- b- **La classe Jeu** :
  - C'est la classe principale du projet, définissant le squelette du jeu. Elle est abstraite et fournit des méthodes génériques telles que `appliqueRegles`, `terminePartie`, et `demarrePartie`, qui sont destinées à être mises en œuvre par les sous-classes spécifiques du jeu.
- c- **La classe LanceurDeJeu** :

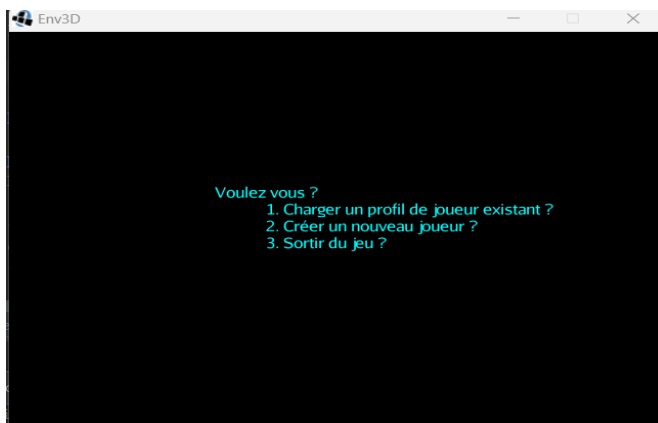
- Cette classe contient la méthode main et sert de point d'entrée pour le jeu. Elle instancie la classe Jeu et lance le déroulement du jeu.
- d- **Les classes spécifiques du jeu (Chronometre, Dico, EnvText, EnvTextMap, Partie, Profil, Room, Tux, etc...) :**
- Ces classes sont des composants spécifiques du jeu qui sont utilisés par la classe Jeu pour créer, gérer et afficher différents aspects du jeu. Par exemple, Tux représente le personnage du jeu, Profil gère les données du joueur, et Partie représente une instance de jeu spécifique.

2 – **Le répertoire xml** : intégré dans le répertoire source du projet, joue un rôle central dans la gestion des données du jeu Tux LetterGame. Cette section du projet est soigneusement organisée pour accueillir des fichiers XML, XSD et XSL, reflétant une approche méthodique de la gestion des données.

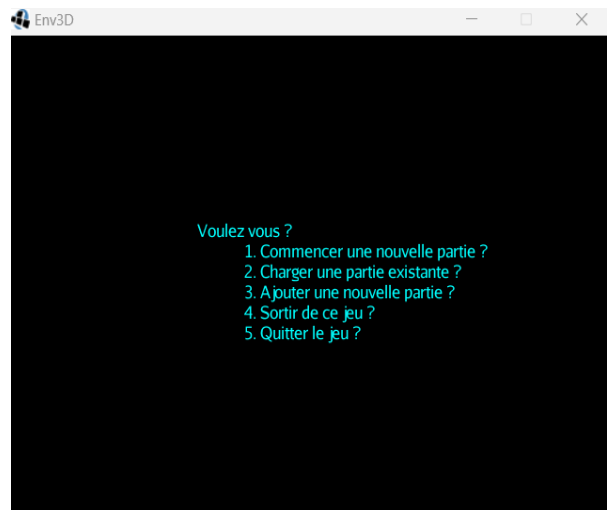
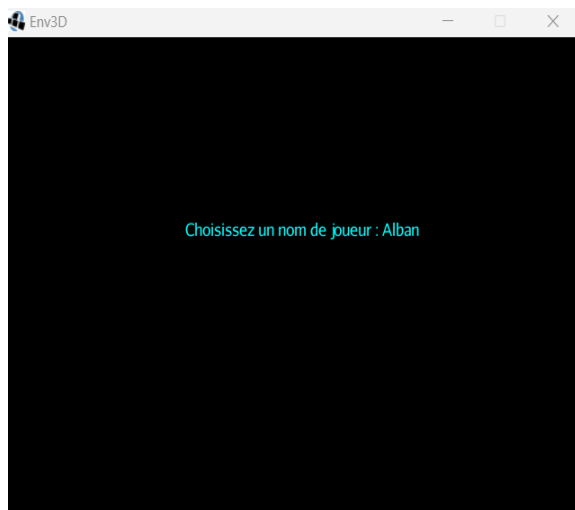
## IV- **Fonctionnalités du jeu**

Les fonctionnalités clés qui définissent l'expérience de jeu comprennent :

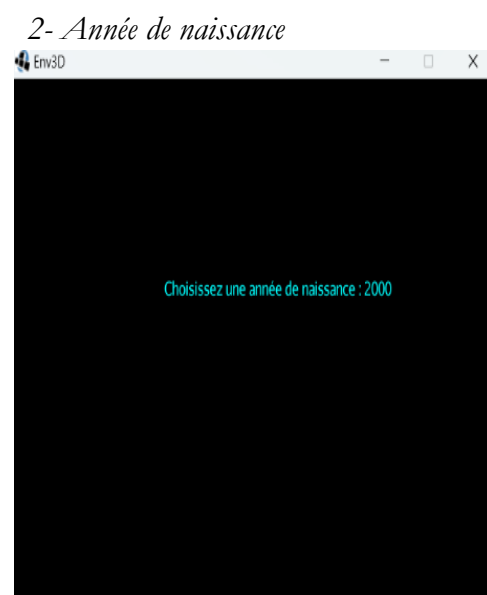
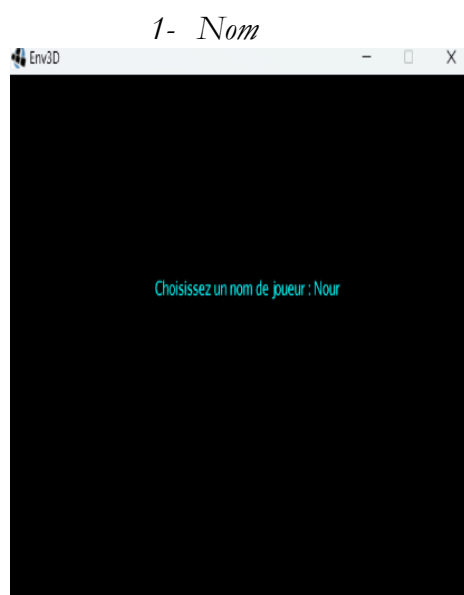
### 1- **Menu principal**



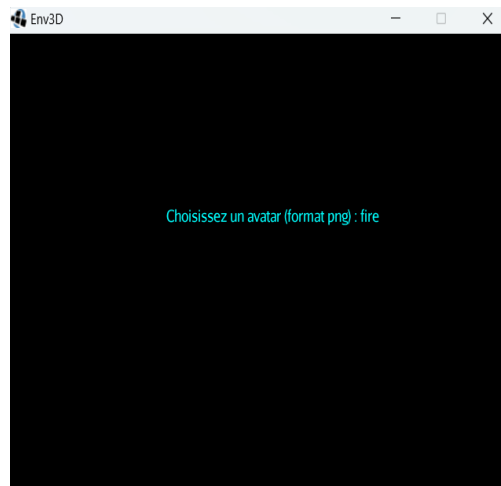
- 1- **Chargement d'un profil de joueur existant** : Quand le joueur décide de charger un profil de joueur existant, on lui demande de saisir le nom du joueur (fonction ***getNomJoueur() : String*** de la classe **Jeu.java**). Si le profil existe, le menu de jeu est affiché. Si non, retour sur le menu principal. Cette vérification est faite par la fonction ***charge(nom : String) : boolean*** de la classe **Profil.java**



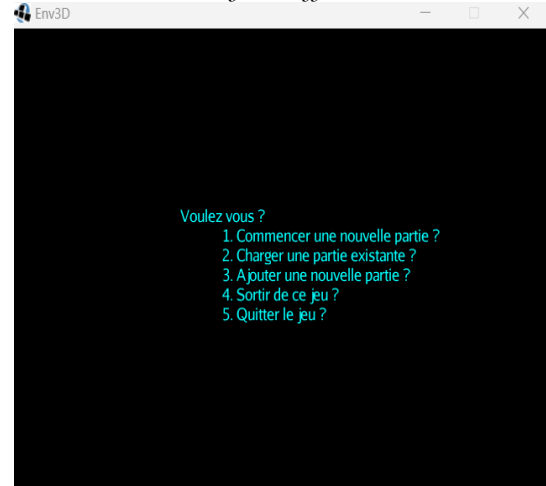
- 2- Création d'un nouveau joueur avec nom, date de naissance et avatar : Si le joueur choisi de créer un nouveau profil, le jeu lui demande son nom (fonction ***getNomJoueur() : String*** de la classe **Jeu.java**), son année de naissance (fonction ***getDateNaissance() : String*** de la classe **Jeu.java**) et son avatar (fonction ***getAvatar() : String*** de la classe **Jeu.java**) et lui affiche le menu Jeu.



### 3 – Avatar

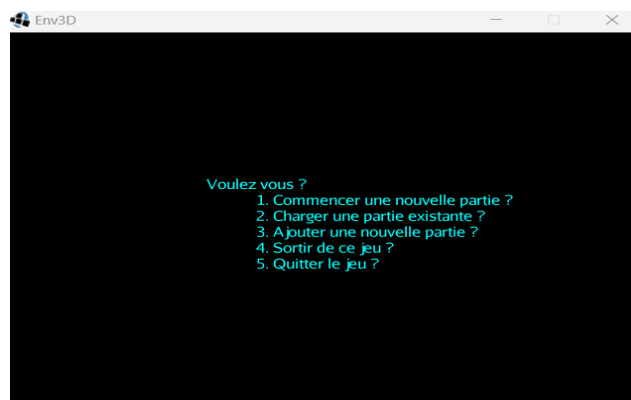


### le menu du jeu s'affiche



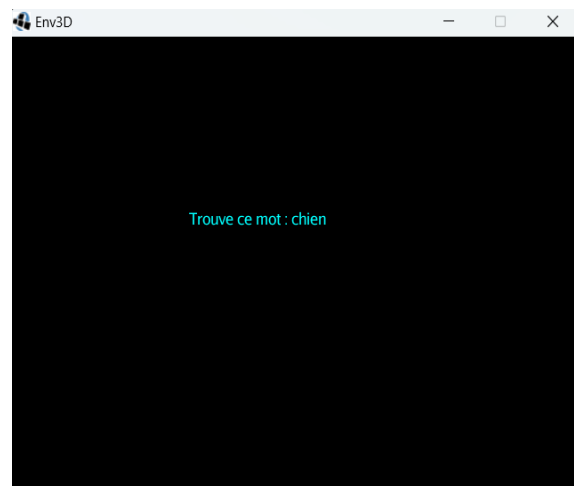
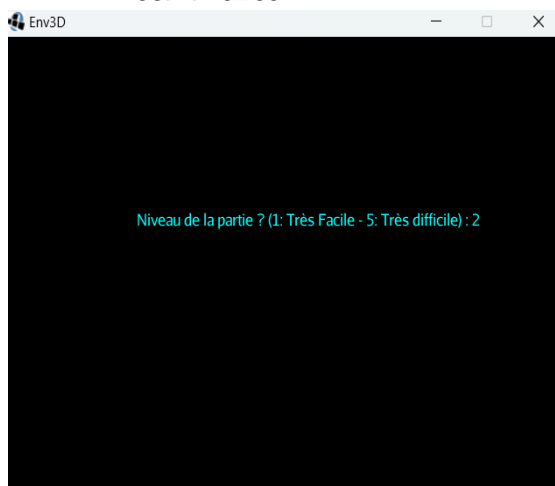
3- Sortir du jeu : le jeu fini et l'interface disparaît.

## 2- Menu de Jeu



1- Lancement d'une nouvelle partie :

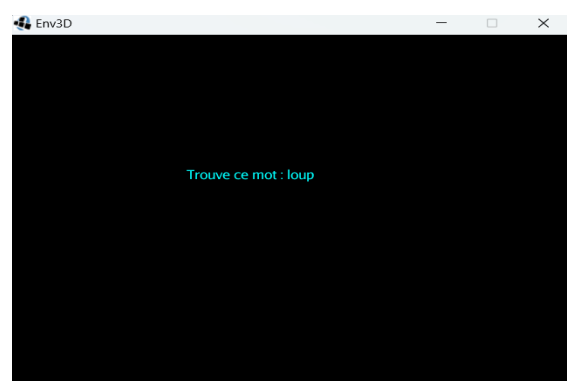
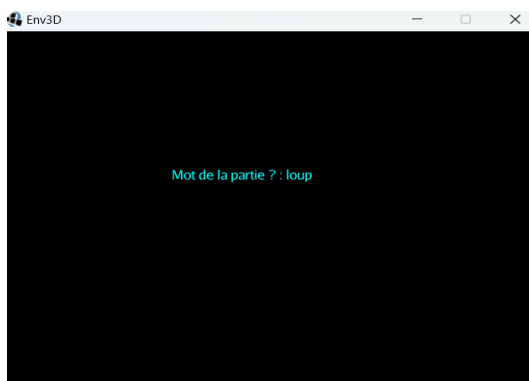
- Un niveau de difficulté est demandé au joueur
- Un mot correspondant à ce niveau de difficulté est chargé et la partie commence.



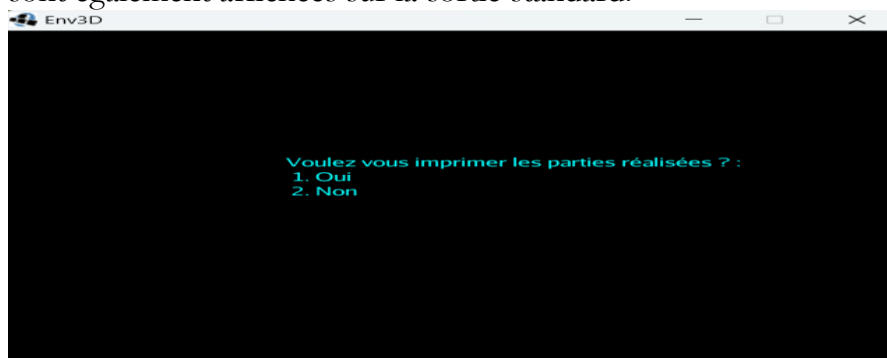




- 2- Chargement d'une partie existante : Si le joueur choisi de charger une partie existante, le jeu lui demande le mot correspondant à la partie (fonction ***getMotPartie() : String*** de la classe ***Jeu.java***) . Si le mot existe alors la partie lui est chargée. Sinon, le joueur est redirigé vers le menu Jeu



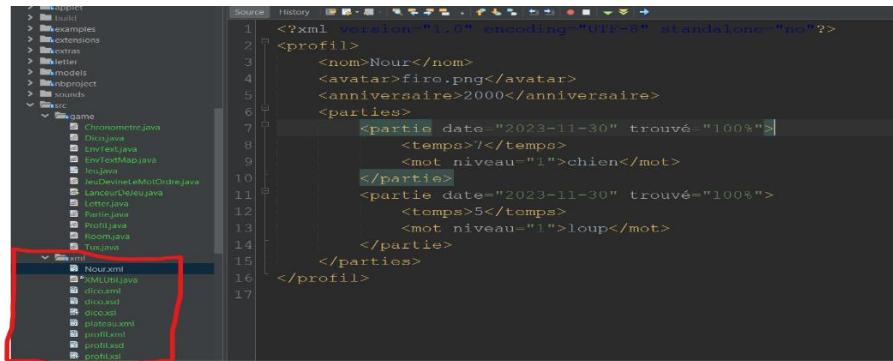
- 3- Ajout d'une nouvelle partie : Permet au joueur d'ajouter son propre mot pour pouvoir deviner les lettres (même scénario d'image que le chargement d'une partie existante mais il s'agit d'un mot choisit par le joueur lui-même) . Une fois le mot ajouté, une écriture DOM est faite en appelant la fonction ***ajouterMot(mot : String, niveau : int) : void*** de la classe **Dico.java** qui ajoute le mot et son niveau associé dans le dictionnaire en éditant le document DOM représentant le dictionnaire.
- 4- Sortie de la partie en cours : A cette étape, un menu d'impression est proposé au joueur(fonction ***getImprimeParties() : String*** de la classe **Jeu.java**) . S'il choisit d'imprimer sa partie, un fichier XML est automatiquement créé via une écriture DOM et nommé d'après le nom du joueur(fonction ***sauvegarderNewProfil(filename :String) : void*** de la classe **Profil.java** s'il s'agit d'un nouveau joueur ou la fonction ***sauvegarder(filename :String) : void*** de la classe **Profil.java** s'il s'agit d'un profil de joueur existant) . Ce fichier contient les détails des parties jouées par l'utilisateur. En outre, les informations sont également affichées sur la sortie standard.



```

-----Partie jouée-----
Date : 30/11/2023
Mot à trouver : chien
Niveau : 1
Trouvé : 100%
Temps mis : 7secondes
-----
-----Partie jouée-----
Date : 30/11/2023
Mot à trouver : loup
Niveau : 1
Trouvé : 100%
Temps mis : 5secondes
-----
AL lib: alc_cleanup: 1 device not closed
BUILD SUCCESSFUL (total time: 1 minute 16 seconds)

```



5- Quitter le jeu : le jeu fini et l'interface disparaît.

### 1- Sélection du niveau de difficulté

Lorsque le joueur opte pour le lancement d'une nouvelle partie ou l'ajout d'une partie personnalisée, le menu de sélection du niveau de difficulté s'affiche. Le joueur est ainsi en mesure de définir le niveau de complexité du jeu, échelonné de 1, représentant une difficulté très facile, à 5, correspondant à une difficulté très élevée.

### 2- Chargement de mots

Le jeu utilise le fichier **dico.xml** pour charger dynamiquement des mots adaptés au niveau de difficulté sélectionné par le joueur.

Deux options de récupération des mots depuis le fichier **dico.xml** :

- 1- Lecture DOM du fichier effectuée par la fonction ***lireDictionnaireDOM(path : String, filename :String) : void*** de la classe **Dico.java**
- 2- Lecture SAX du fichier effectuée par la fonction ***lireDictionnaire() : void*** de la classe **Dico.java**

Dans les deux cas, les mots lus sont dynamiquement ajoutés aux listes de niveau correspondant en utilisant la fonction ***ajouteMotADico(niveau : int, mot : String) : void*** de la classe **Dico.java**.

### 3- Déplacement du Personnage Tux

Contrôle des déplacements du Tux permettant sa navigation dans l'environnement grâce à la fonction ***deplace() : void*** de la classe **Tux.java** qui permet d'ajuster les coordonnées  $x$ ,  $y$  et  $z$  du personnage Tux.

### 4- Détection des lettres par Tux

Lorsque Tux se déplace dans l'environnement, la fonction ***tuxTrouveLetter(indice : int) : boolean*** de la classe **JeuDevineLeMotOrdre.java** permet de vérifier si Tux a détecté la bonne lettre lorsqu'une collision se produit. Elle utilise la fonction

***collision(letter : Letter) : boolean*** de la classe **Jeu.java** pour savoir s'il y'a collision ou pas. S'il y a collision avec la bonne lettre alors cette dernière est supprimée de l'environnement.

## 5- Fin d'une partie

La fin d'une partie est déclenchée par deux conditions distinctes :

- 1- **Écoulement du temps** : La première condition de la fin d'une partie se déclenche lorsque le temps imparti pour la recherche des lettres s'est écoulé. Cette vérification est effectuée par la fonction ***appliqueRegles(partie: Partie): void*** de la classe **JeuDevineLeMotOrdre.java**, utilisant la fonction ***reainsTime(): boolean*** de la classe **Chronometre.java**.
- 2- **Détection de toutes les lettres par Tux** : La seconde condition intervient dès que Tux parvient à détecter toutes les lettres présentes dans l'environnement du jeu.

Dans les deux cas deux fonctions de la classe **JeuDevineLeMotOrdre.java** sont utilisées :

- 1- La fonction ***appliqueRegles(partie: Partie): void*** qui permet de vérifier le temps restant (fonction ***reainsTime(): boolean*** de la classe **Chronometre.java**) et le nombre de lettres restantes (en vérifiant la valeur de l'attribut **nbLettresRestantes**) ;
- 2- La fonction ***terminePartie(partie : Partie) : void*** qui permet d'arrêter le chronomètre (grâce à la fonction ***stop() : void*** de la classe **Chronometre.java**), d'enregistrer le temps mis pour collecter les lettres (grâce à la fonction ***setTemps(temps : int) : void*** de la classe **Partie.java**), et d'enregistrer le pourcentage des lettres trouvées (grâce à la fonction ***setTrouve(nbLettresRestantes : int) : void*** de la classe **Partie.java**).

## CONCLUSION

En résumé, la réalisation du projet Tux LetterGame a été une aventure passionnante où nous avons pu explorer et maîtriser les tenants et aboutissants des technologies **XML**, **XSLT** et **XSD**. De la conception à la mise en œuvre pratique en utilisant **les Parsers DOM et SAX en Java**, cette expérience nous a permis de consolider nos connaissances et de mettre en pratique les compétences acquises. La formalisation des données, avec un accent particulier sur la manipulation de fichiers XML, a été un aspect clé du projet, nous dotant d'une compréhension concrète et applicable des technologies XML.