

A dark blue vertical bar runs along the left edge of the page. A blue arrow-shaped banner points to the right from this bar, containing the date. In the bottom-left corner, there are several thin, curved, light blue lines that sweep upwards and to the right.

06/12/2022

# FORMALISATION DE DONNEES – TECHNOLOGIES XML

Rapport du projet TuxLetterGame

Zeïnabou Bissi KONE – Sicong XU  
L3 MIAGE - UGA

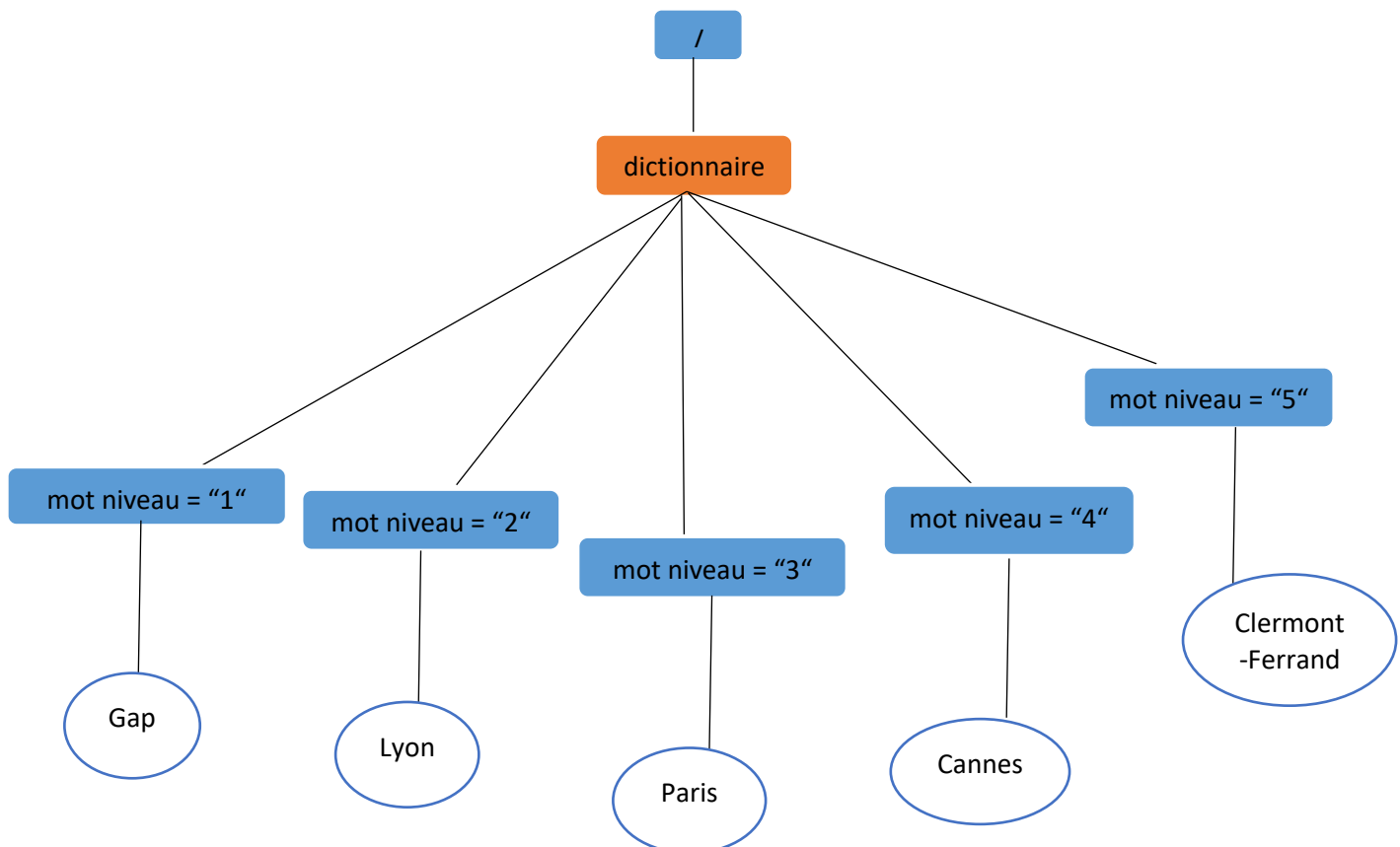
## I. INTRODUCTION

Ce présent rapport décrit l'ensemble des réalisations effectuées au cours du projet **Tux Letter Game** de l'UE **FDD-XML**. Tout au long de ce projet, nous avons mis en pratique nos acquis théoriques sur les fichiers **xml**, les **schémas xml**, les transformations **xlst** et une grande partie sur la création d'un mini-jeu 3D pour approfondir nos acquis précédemment cités mais également celles portant sur les **parseurs**. Il a été réalisé de façon incrémentale et nous allons détailler, ci-après l'ensemble des réalisations faites.

## II. REALISATIONS

### a. XML

Cette première partie consistait à mettre en pratique nos savoirs sur le langage **xml**, qui représente la base de notre jeu. Nous avons écrit un fichier *dico.xml* qui est en quelque sorte un « dictionnaire » de mots qui sont identifiés par des niveaux allant de 1 à 5. Nous avons choisi comme mots des noms de villes françaises car nous trouvions cela assez original. L'arbre XML peut être représenté comme suit :



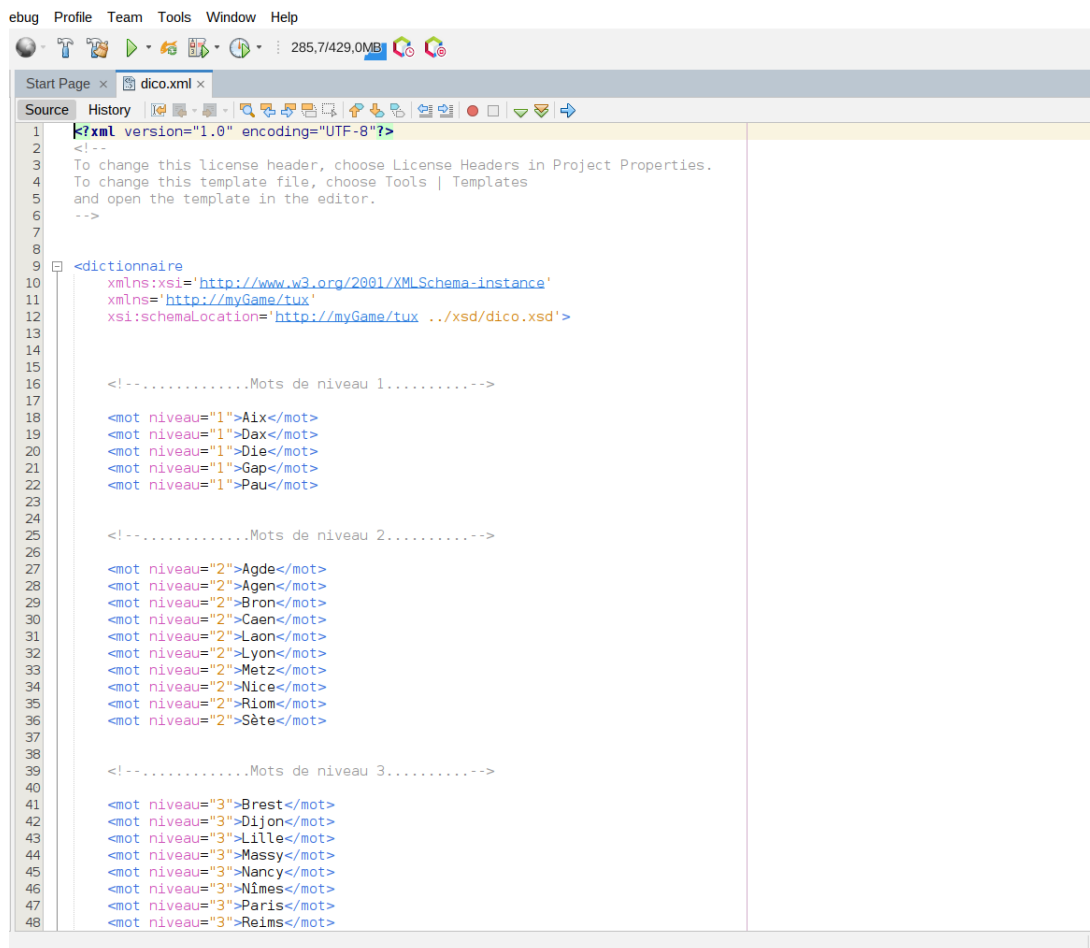
ABRE XML DU DICO

Dans notre dico.xml, la longueur des mots commence par 3. Pour avoir un dictionnaire un peu plus harmonieux et cohérent, nous avons choisi les mots de telle sorte que :

- Les noms de villes composés de 3 lettres sont des mots du niveau 1
- Les noms de villes composés de 4 lettres sont des mots du niveau 2
- Les noms de villes composés de 5 lettres sont des mots du niveau 3
- Les noms de villes composés de 6 lettres sont des mots du niveau 4
- Les noms de villes composés de 7 lettres et plus sont des mots du niveau 5

Les niveaux ici sont donc des **attributs** des éléments mot.

Un petit aperçu de notre fichier se trouve sur l'image ci-dessous.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 To change this license header, choose License Headers in Project Properties.
4 To change this template file, choose Tools | Templates
5 and open the template in the editor.
6 -->
7
8
9 <dictioinaire
10   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
11   xmlns='http://myGame/tux'
12   xsi:schemaLocation='http://myGame/tux ../xsd/dico.xsd'>
13
14
15   <!--.....Mots de niveau 1.....-->
16
17   <mot niveau="1">Aix</mot>
18   <mot niveau="1">Dax</mot>
19   <mot niveau="1">Die</mot>
20   <mot niveau="1">Gap</mot>
21   <mot niveau="1">Pau</mot>
22
23   <!--.....Mots de niveau 2.....-->
24
25   <mot niveau="2">Agde</mot>
26   <mot niveau="2">Agen</mot>
27   <mot niveau="2">Bron</mot>
28   <mot niveau="2">Caen</mot>
29   <mot niveau="2">Laon</mot>
30   <mot niveau="2">Lyon</mot>
31   <mot niveau="2">Metz</mot>
32   <mot niveau="2">Nice</mot>
33   <mot niveau="2">Riom</mot>
34   <mot niveau="2">Sète</mot>
35
36   <!--.....Mots de niveau 3.....-->
37
38   <mot niveau="3">Brest</mot>
39   <mot niveau="3">Dijon</mot>
40   <mot niveau="3">Lille</mot>
41   <mot niveau="3">Massy</mot>
42   <mot niveau="3">Nancy</mot>
43   <mot niveau="3">Nîmes</mot>
44   <mot niveau="3">Paris</mot>
45   <mot niveau="3">Reims</mot>
```

## b. XMLSCHEMA

Dans cette partie, nous avons écrit les *schémas xml* correspondant aux instances xml du dictionnaire (*dico.xml*) et du profil (*profil.xml*) qui nous a été fourni). Nous avons donc correctement contraint ces mots (absence de chiffres, impossibilité d'écrire des mots composés sans tirets et des caractères spéciaux comme @, #, \*, etc.). Ce premier schéma xml de dico.xml est écrit dans le fichier *dico.xsd*. Pour prouver l'efficacité de notre schéma, nous avons enrichi davantage notre dictionnaire (*dico.xml*) avec des noms de villes composés, qui présentent des lettres assez spéciales, etc.

Nous avons également écrit le schéma xml du profil (*profil.xsd*). Pour ces deux schémas, nous utilisons l'espace de nom <http://myGame/tux>.

Les fichiers *dico.xml* et *profil.xml* se trouvent dans le sous-répertoire **xml** du répertoire **Data** et les fichiers *dico.xsd* et *profil.xsd* dans le sous-répertoire **xsd**.

## c. XSLT

Dans cette partie, nous avons écrit des feuilles de transformation **xslt** permettant de générer des page **html** simple afin d'afficher notre dictionnaire et notre profil.

Pour commencer, nous avons écrit une première feuille xslt nommée *dico1.xsl* qui permettrait un affichage des mots du dictionnaire **triés par ordre alphabétique**.

Ensuite, nous avons écrit une deuxième feuille xslt nommée *dico2.xsl* qui permettrait d'afficher les mots du dictionnaire par **ordre niveau croissant et par ordre alphabétique**.

Enfin, nous avons écrit une feuille xslt nommée *profil.xsl* qui permettrait d'afficher le profil du joueur.

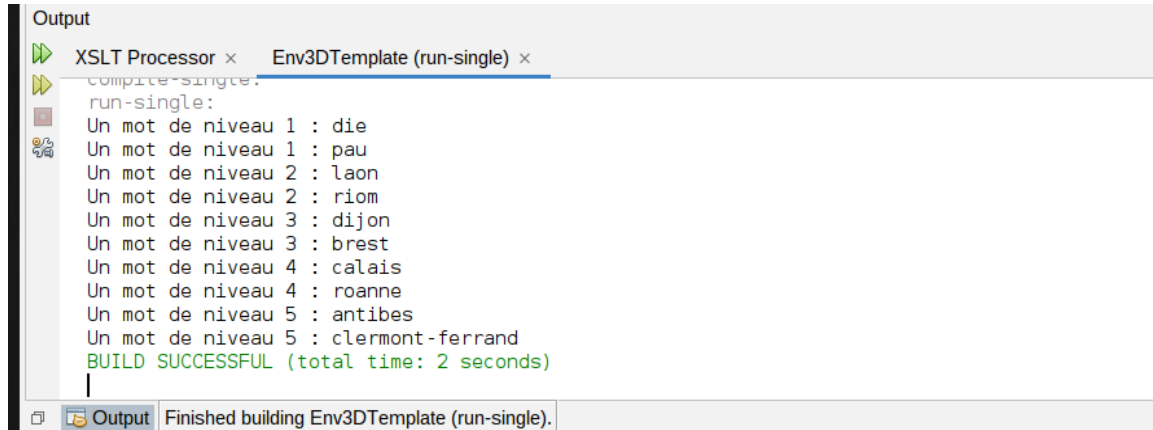
Les pages qui ont été générées grâce à ces feuilles se trouvent dans le dossier **html** avec *dico1.html* pour le **dico1.xsl** et *dico2.html* pour le **dico2.xsl**. Il s'y trouve également le fichier *profil.html*, qui a été généré par *profil.xsl*. Les feuilles de style utilisées pour la mise en page se trouvent quant à elle, dans le dossier **css** et les éventuelles images utilisées pour les backgrounds, etc. sont le répertoire **images**.

## d. TUX – MINI JEU 3D

Ici, nous avons globalement mis en place un jeu de devinette de mots. Le joueur va devoir deviner des mots de notre *dico.xml*. Nous avons donc mis en avant nos acquis sur les **parseurs** et la programmation objet Java.

## Parsing du dico :

Nous avons utilisé le parseur DOM pour récupérer des mots du dictionnaire en fonction de leurs niveaux. Le fichier Dico.java du répertoire **game** contient l'ensemble des méthodes nécessaires pour faire ceci et le fichier TestDico.java (du répertoire **test**) nous permet de l'exécuter. Un petit aperçu du résultat obtenu se trouve ci-dessous :



```
Output
XSLT Processor x Env3DTemplate (run-single) x
compute-singte.
run-single:
Un mot de niveau 1 : die
Un mot de niveau 1 : pau
Un mot de niveau 2 : laon
Un mot de niveau 2 : riom
Un mot de niveau 3 : dijon
Un mot de niveau 3 : brest
Un mot de niveau 4 : calais
Un mot de niveau 4 : roanne
Un mot de niveau 5 : antibes
Un mot de niveau 5 : clermont-ferrand
BUILD SUCCESSFUL (total time: 2 seconds)
```

## Le jeu

Par la suite, nous avons écrit des programmes java qui nous permettraient de mettre en place l'environnement du jeu et le jeu en lui-même. A ce niveau, nous avons été un peu contraintes par le temps et nous n'avons pas pu réaliser tout ce que nous aurions voulu faire. Néanmoins, nous avons pu avoir un jeu qui fonctionne et qui donne un fichier xml du profil du joueur.

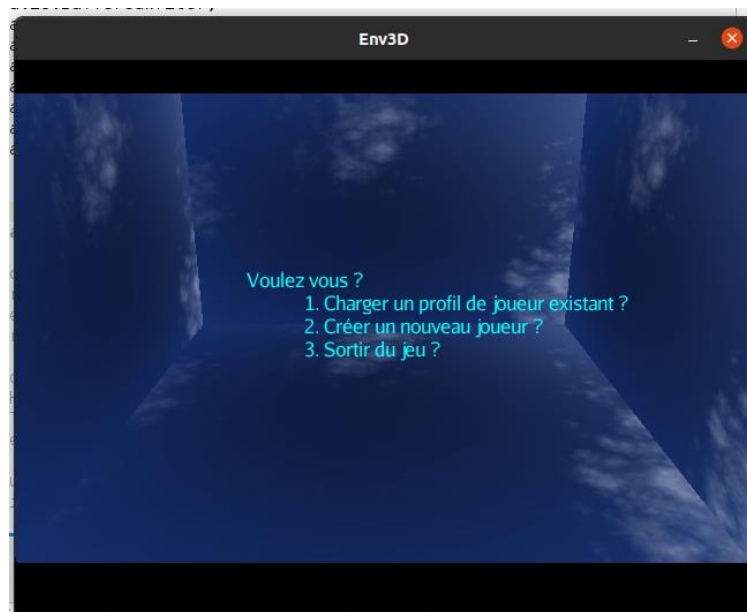
### Règles du jeu :

Lors du lancement du jeu, le joueur doit faire le choix entre :

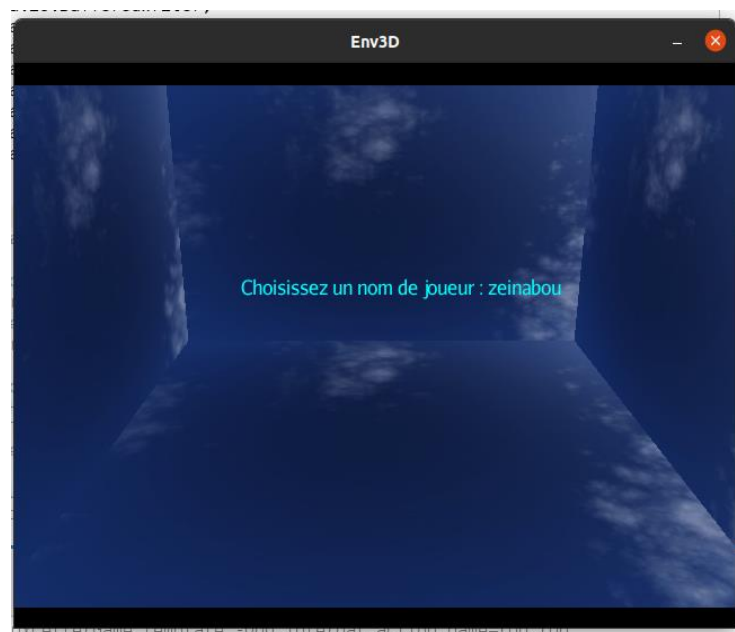
1. Charger le profil d'un joueur existant (fonctionnalité qui ne fonctionne malheureusement pas) ;
2. Créer un nouveau joueur ;
3. Sortir du jeu ;

Malheureusement, nous n'avons pas eu le temps de parfaire le chargement d'un profil existant.

Les deux saisies qui pourraient donc aboutir son la « 2 » et la « 3 ».



Si le joueur souhaite sortir du jeu, il saisira « 3 » et en sortira définitivement ; sinon, lorsqu'il aura saisi « 2 », il verra apparaître un deuxième menu qui lui indiquera de saisir son nom. Sur cet exemple, le joueur s'appelle « zeinabou » :

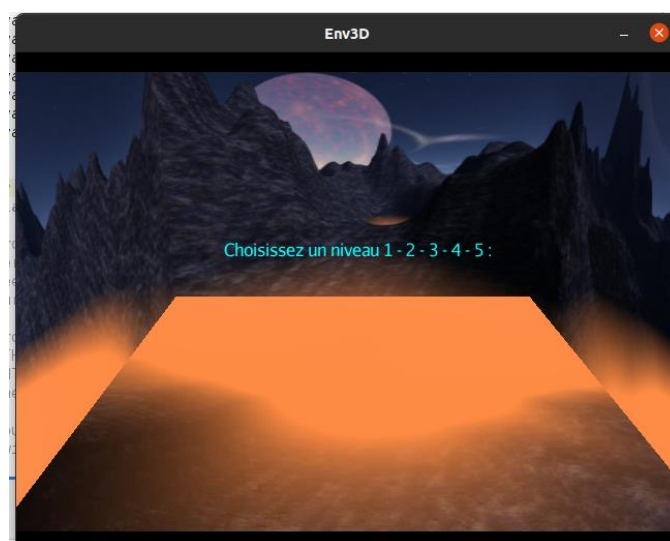


Lorsqu'il aura fini de saisir son nom, il doit cliquer sur la toute **Entrée** du clavier. Il sera ainsi redirigé vers un troisième menu qui doit lui permettre de pouvoir soit :

1. Commencer une nouvelle partie
2. Charger une partie existante
3. Sortir du jeu (pour revenir au premier menu)
4. Quitter le jeu (pour sortir complètement du jeu)

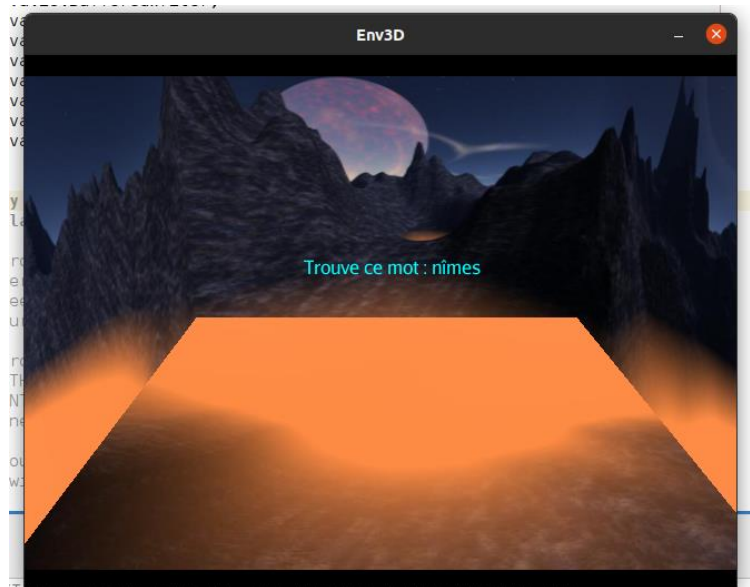


Si le joueur saisi « 1 » au clavier, il sera donc demandé de choisir le niveau du mot qu'il souhaiterait deviner (entre 1 et 5). Ceci se passe dans l'environnement comme l'illustre l'image ci-dessous.



- Si le joueur saisit un chiffre dans l'intervalle  $[1,5]$  et qu'il aura cliqué sur **Entrée**, un mot du dictionnaire de ce niveau lui sera donné pour qu'il le devine ;
- S'il saisit cependant un chiffre autre que  $\{1, 2, 3, 4, 5\}$ , il sera demandé de deviner un mot du niveau 1 par défaut (étant donné que c'est le plus facile).

Pour poursuivre notre exemple , le niveau choisi par ce joueur est le 3. Le mot qu'il devra deviner va donc apparaître dans l'environnement pendant 5 secondes avant qu'il ne joue.

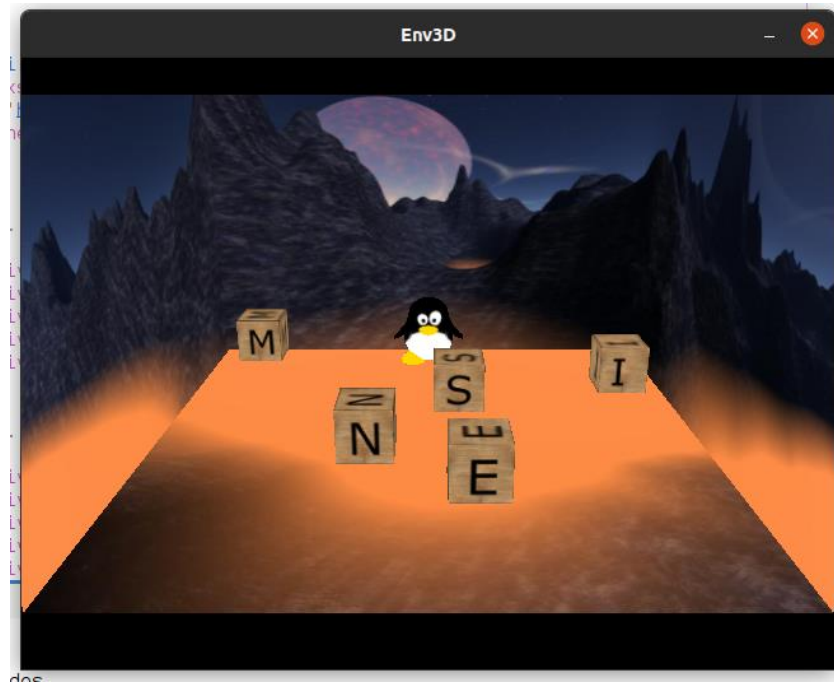


Ici, le mot à deviner est nîmes. Vous pourrez remarquer que le même mot est affiché dans la console :





Lorsque les 5 secondes seront passées, les lettres du mot apparaissent dans l'environnement et tux est prêt à jouer 😊.



A partir de là, il doit maintenant essayer de trouver les lettres de ce mot dans l'ordre, soit : **n** – **i** – **m** – **e** – **s**.

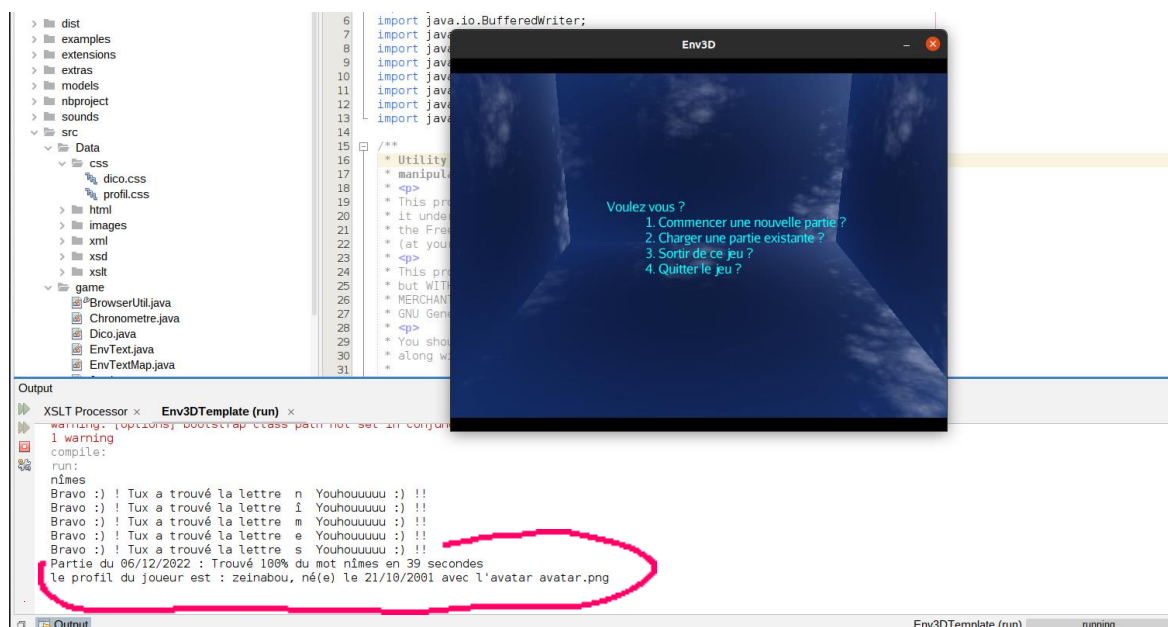
A chaque fois qu'il trouve la lettre attendue, on fait un affichage dans la console dans ce sens (voir schéma ci-dessous) et cette dernière disparaît de l'environnement.

```
Output
XSLT Processor x Env3DTemplate (run) x
Deleting: /home/zeinabou/NetBeansProjects/FDD_XML/Template/TuxLetterGame_template/build/built-jar.properties
deps-jar:
Updating property file: /home/zeinabou/NetBeansProjects/FDD_XML/Template/TuxLetterGame_template/build/built-jar.properties
Compiling 3 source files to /home/zeinabou/NetBeansProjects/FDD_XML/Template/TuxLetterGame_template/build/classes
warning: [options] bootstrap class path not set in conjunction with -source 1.6
1 warning
compile:
run:
rimes
Bravo :) ! Tux a trouvé la lettre n Youhouuuuuu :) !!
Bravo :) ! Tux a trouvé la lettre i Youhouuuuuu :) !!
Bravo :) ! Tux a trouvé la lettre m Youhouuuuuu :) !!
Output Env3DTemplate (run) running...
```

Lorsqu'il aura trouvé toutes les lettres, le joueur doit appuyer sur la touche **Echap** pour mettre fin à la partie. Ici, nous avons tenté de mettre une fonctionnalité qui permettrait au tux de sortir du jeu dès lors qu'il aura deviné le mot, mais malheureusement elle n'était pas fonctionnelle et nous avons préféré améliorer ce que nous avons pour ne pas perdre de temps et ne pas rendre un truc assez propre à la fin.

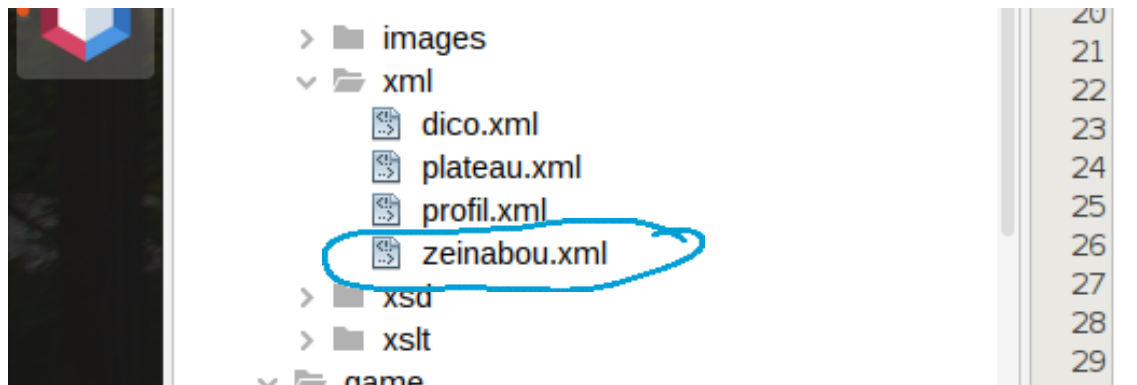
Après avoir cliqué sur la touche **echap**, le récapitulatif de la partie et du profil du joueur s'affichent dans la console et le menu du joueur réapparaît.

Le récapitulatif de la partie donne le **pourcentage des lettres du mot trouvées et le temps mis pour faire la partie.**

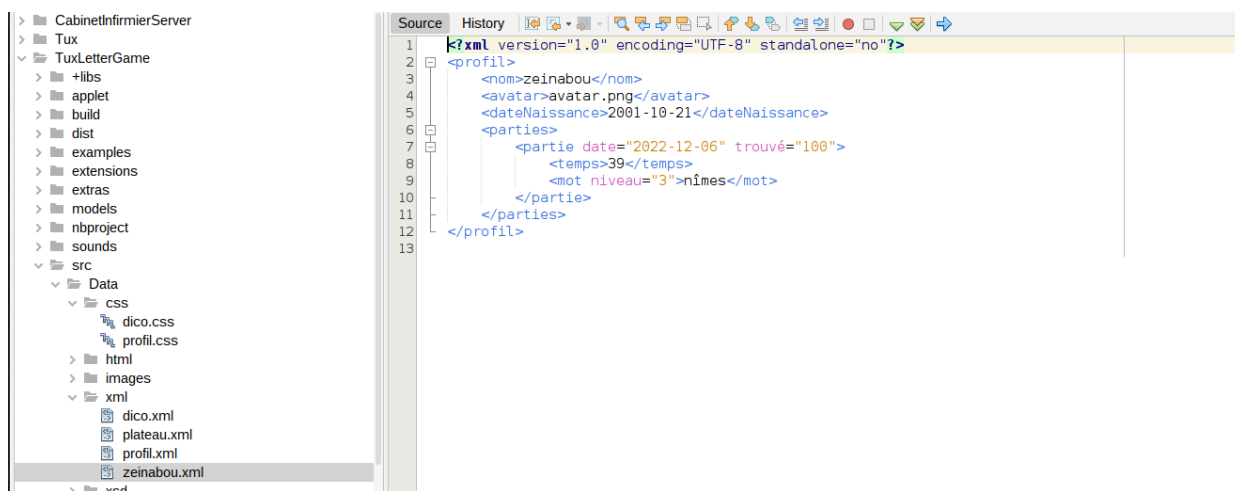


A ce stade, le joueur a la possibilité de recommencer une nouvelle partie s'il le désire. Mais pour cet exemple, nous nous sommes limités à une partie. Et donc, le joueur saisit « 4 » pour sortir définitivement du jeu.

Lorsque cela est fait, dans le dossier xml, on peut remarquer que le fichier profil du joueur est créé et se nomme **zeinabou.xml**.



Ce document xml contiendra les éléments suivants :



Pour une vue et une analyse un peu plus poussée de notre réalisation, le fichier [siconq.xml](#) est le profil du joueur « **siconq** » qui a joué plusieurs parties.

### III. CONCLUSION - POUR ALLER PLUS LOIN

Nous sommes assez contentes de nos réalisations. La partie **XML/XSD/XSLT** a bien été réalisée. En ce qui concerne la partie **JAVA**, nous nous sommes également bien débrouillées avec le parseur DOM qui nous a permis de pouvoir lire le fichier dico.xml et faire en sorte que le profil du joueur puisse être créé à la fin de son jeu. Nous avons également voulu améliorer le menu afin de permettre au joueur de saisir sa date de naissance mais malheureusement la code SLASH de Keyboard ne permettait pas cela. De même pour l'avatar ( car pas de constructeur qui le prenne en paramètre).

Pour aller plus loin si nous avions plus de temps, nous aurions voulu :

- Faire fonctionner la fonctionnalité du chargement du profil d'un joueur existant ;

- Pouvoir améliorer le menu ;
- Pouvoir faire gérer la sortie du jeu par tux lui-même ;
- Donner la possibilité au joueur de saisir sa date de naissance et son avatar (en créant un autre constructeur ;

Cependant, nous avons respecté les diagrammes UML des classes qui nous ont été fournies, et c'est pour cela que nous n'avons pas rajouter des constructeurs, méthodes...