Rapport de projet
Tux Letter Game

UE Formalisation Des Données - Technologies XML
L3 MIAGE

FRANCES Tom
BOU SERHAL Jean
GOUDON Justin

Table des matières

I. Introduction

Le projet que nous avions à réaliser cette année était un mini-jeu en 3D permettant d'appliquer toutes les notions de la programmation vues au cours de ce premier semestre.

Comme demandé, nous avons réalisé les fonctionnalités de base du jeu, et nous y avons apporté quelques modifications personnelles que nous trouvions intéressantes.

Consignes de préparation :

Avant de pouvoir lancer le jeu, vous devez :

- remplacer le répertoire **TuxLetterGame_template/src** par le répertoire **src** présent dans l'archive zip (à dézipper)
- ajouter les images de lettres présentes dans l'archive zip **lettresBonus.zip** au répertoire **TuxLetterGame_template/models/letter**. Ce répertoire contient plusieurs images de lettres accentuées, tirets, etc, ajoutées pour pouvoir utiliser les mots contenant des accents sans ambiguïté.
- pour la création d'un profil, le nom de l'avatar n'est pas encapsulé (manque de temps) : vous devez choisir entre les noms suivants : naruto, cinder, archer, sheldon et morty.

Tux est un jeu dans lequel nous devons déplacer un personnage et récupérer des lettres pour pouvoir reformer un mot dans le temps imparti, en utilisant les commandes suivantes :

- Flèches de gauche et droite : déplacements horizontaux
- Flèches du haut et du bas : déplacements verticaux

Le mot que nous allons essayer de retrouver au cours d'une partie est choisi de façon aléatoire dans un dictionnaire, en fonction du niveau choisi par l'utilisateur.

Lors de l'identification de l'utilisateur, son profil est enregistré. Il pourra donc se reconnecter et jouer avec son profil. C'est pour cela que dans le menu principal, il peut :

- Charger un profil existant
- Créer un nouveau profil
- Quitter le jeu

Nous avons décidé de classer les mots en 6 différents niveaux :

- <u>Niveau 1 :</u>
  Le mot est entièrement donné avant le début de la partie, il est affiché pendant 5 secondes. Le joueur dispose de 20 secondes pour trouver le mot.
  Le mot fait de 3 à 5 lettres, et est un mot courant de la langue française.
- <u>Niveau 2 :</u>
  Le mot est entièrement donné avant le début de la partie, il est affiché pendant 5 secondes. Le joueur dispose de 23 secondes pour trouver le mot.
  Le mot fait de 5 à 8 lettres, et est un mot courant de la langue française.

- Niveau 3 :

  Le mot est entièrement donné avant le début de la partie, il est affiché pendant 5 secondes. Le joueur dispose de 25 secondes pour trouver le mot.
  Le mot fait de 9 à 11 lettres et est un mot un peu moins courant que les mots des 3 premiers niveaux.

Dans ces 3 premiers niveaux, les lettres sont récupérables uniquement dans le bon ordre, il est par conséquent impossible de faire une erreur dans le mot (on peut néanmoins ne pas retrouver le mot dans son intégralité avant la fin du temps imparti).

- Niveau 4 :

  Le mot est entièrement donné avant le début de la partie, il est affiché pendant 5 secondes. Le joueur dispose de 30 secondes pour trouver le mot.
  Le mot fait de 7 à 11 lettres, est un mot un peu moins courant de la langue française.

- Niveau 5 :

  Le mot est entièrement donné avant le début de la partie, il est affiché pendant 5 secondes. Le joueur dispose de 45 secondes pour trouver le mot.
  Le mot fait de 8 à 14 lettres, est un mot très peu utilisé dans la langue française (pour les plus littéraires d'entre nous).

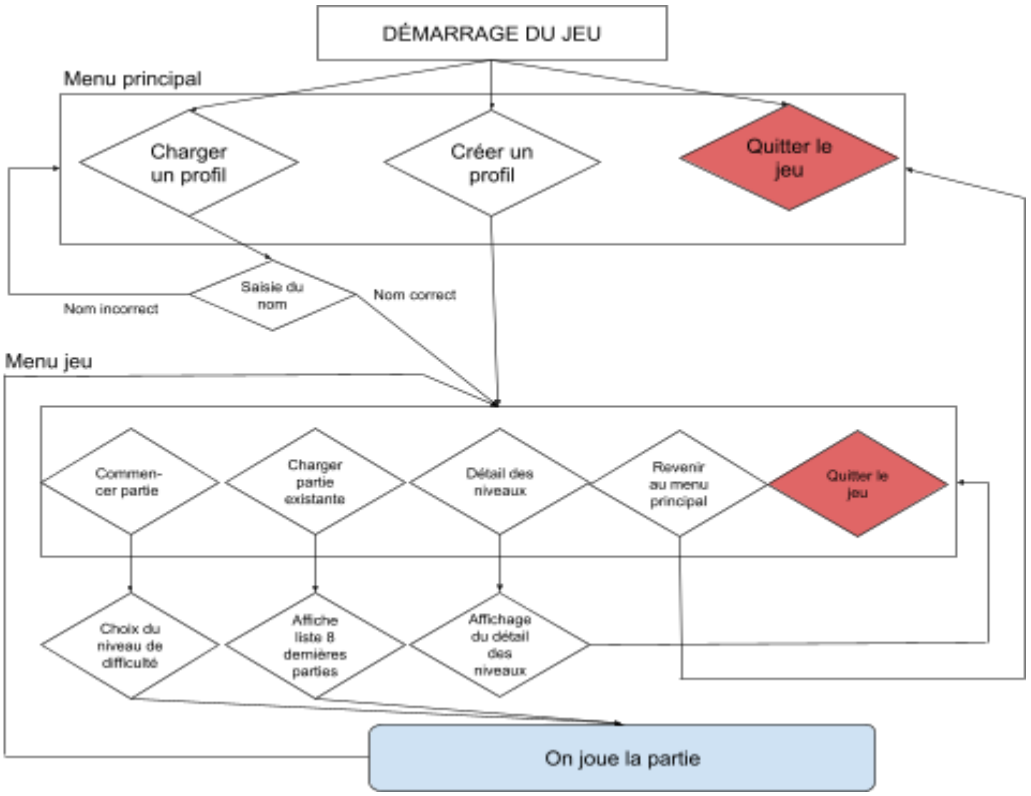
- Niveau 6 :

  On ne donne que la première lettre et la dernière du mot, et le nombre de lettres, ces informations sont affichées pendant 5 secondes. Le joueur dispose de 60 secondes pour trouver le mot.
  Le mot fait de 6 à 8 lettres, et est un mot relativement simple et courant de la langue française.

Dans ces 3 derniers niveaux, les lettres sont récupérables dans n'importe quel ordre, des erreurs sont donc possibles, et seront comptabilisées dans le score.

Une fois que l'utilisateur a créé ou chargé son profil, il peut commencer une nouvelle partie, en charger une ancienne, ou quitter le jeu. Dans le cas du chargement d'une partie, l'utilisateur verra ses 8 dernières parties non terminées, et pourra choisir l'une d'entre elles et la rejouer.
La partie se termine lorsque le temps est écoulé, ou lorsque toutes les lettres ont été récupérées (avec des erreurs ou non). Un message s'affiche nous indiquant si on a gagné ou non. Nous sommes ensuite redirigés sur le menu de jeu pour pouvoir refaire une partie, ou quitter.

II. Organisation du jeu



III. Ajouts et améliorations

- <u>Collision et complétion du mot en fonction de la lettre du cube :</u>

Dans un premier temps, la récupération d'un cube dépendait non pas de la lettre du cube, mais du cube lui-même. Ainsi si un mot comportait plusieurs fois la même lettre, le joueur pouvait atteindre un cube portant la bonne lettre, mais sans parvenir à la récupérer. Nous avons donc amélioré la gestion des collisions entre Tux et les cubes, afin que cette collision tienne compte de la lettre et non du cube.

- <u>Animation de saut de Tux :</u>

Pour rendre le jeu un peu plus dynamique et ludique, nous avons implémenté une animation faisant sauter Tux à la fin d'une partie gagnée par le joueur.

- <u>Fluidité des menus et navigabilité</u>

Comme nous avons plusieurs fois été bloqués lors de la navigation dans nos menus, nous avons dû implémenter des fonctionnalités permettant de toujours retourner au menu d'avant. Nous pouvons donc passer à présent d'un menu à l'autre sans pour autant avoir à redémarrer le jeu. En revanche, dans le menu de sélection de niveau, nous devons obligatoirement sélectionner un niveau, commencer une partie, et la quitter quand elle commence pour retourner au menu de choix de partie. Dans le menu

de sélection de profil, ou de création nous devons terminer de remplir les champs pour retourner au niveau précédent.

- SonarLint

Dans ce projet, nous avons installé l'extension SonarLint pour Netbeans, afin d'améliorer la qualité et la robustesse de notre code. Cela nous a permis de transformer certaines boucles foreach, par exemple *for(Letter l : liste)*, pour finalement utiliser la méthode forEach() appliquée à des collections, plus efficace. Cela nous a également fait afficher nos messages d'erreur ou d'informations de suivi du programme avec un Logger plutôt que des println().

IV. XML et Parsings DOM/SAX

- Dictionnaire

Le dictionnaire est stocké dans un document xml, *dico.xml*. Afin de récupérer les mots qui y sont enregistrés dans notre application java, nous utilisons l'API SAX. Cette API utilise des événements gérés par l'application pour traiter un document xml. Le parseur SAX va parcourir le document xml dans son intégralité, et produira un événement à la rencontre d'un marqueur particulier dont nous aurons défini le traitement dans notre application.
A l'issue de ce parsing, tous les mots et leurs niveaux seront stockés sous forme d'objet dans la mémoire de notre application java, pour pouvoir être utilisés pour lancer les parties.

- Profil

Les profils des joueurs sont stockés dans des documents xml à leurs nom, par exemple *profil_Toto.xml*.
A la création de son profil par un joueur, son document xml est créé si le nom de joueur choisi est disponible (cad si aucun document xml ne porte déjà ce nom). Puis après chaque partie, ces dernières seront elles aussi enregistrées dans le profil xml du joueur.
Pour le chargement d'un profil, l'application va d'abord vérifier si un document xml ayant le nom demandé existe : s'il existe, le profil est chargé et le joueur est amené au menu de jeu, sinon le joueur est renvoyé au menu principal avec un message d'erreur.
Toutes ces opérations sont effectuées par un parser DOM : pour la création et la mise à jour du document xml du profil, les éléments sont créés et écrits dans le document xml par l'application java ; pour le chargement du profil, l'application lit les éléments du document xml par leur nom, et crée les objets correspondant.

- Environnement

Le document *plateau.xml* contient toutes les informations nécessaires à la construction de l'environnement du jeu. Une classe Room implémente le parsing DOM de ces informations afin de récupérer les chemins des images de fond (textures) et les dimensions du plateau.

Au lancement du jeu, le constructeur Room de cette classe est appelé, et par conséquent nous retrouvons les chemins et les dimensions dictées par le fichier xml dans notre jeu.

V. Problèmes rencontrés

a) Lorsque nous avons commencé à implémenter la partie Java de notre jeu et les collisions, nous nous sommes rendus compte qu'il y avait un problème majeur. En effet, si un mot comportait deux lettres identiques, pour réussir la reconstitution du mot, nous ne pouvions pas prendre n'importe laquelle des deux lettres. Cela était dû à l'association des lettres à un indice dans une liste de lettres du mot courant. Par exemple, prenons le mot "Kayak", le deuxième "k" étant associé à l'indice 5 il ne pouvait pas être récupéré par le Tux en première lettre du mot. Pour résoudre ce problème, nous avons implémenté la méthode suivante :

```
protected Letter collision(boolean enOrdre) {
    Letter lettreCollision = null;
    Letter lettre;
    int indiceLettre = 0;
    Iterator<Letter> it = lettresLetter.iterator();

    while (it.hasNext() && lettreCollision == null) {
        lettre = it.next();
        if (distance(lettre) < (tux.getScale() + Letter.LETTER_SCALE) / 1.5) {
            if (!enOrdre) {
                lettreCollision = lettre;
                lettresChar.remove(indiceLettre);
                env.removeObject(lettre);
                lettresLetter.remove(lettre);
            } else if (lettre.getLettre() == lettresChar.get(0)) {
                lettreCollision = lettre;
                lettresChar.remove(0);
                env.removeObject(lettre);
                lettresLetter.remove(lettre);
            }
        }
        indiceLettre++;
    }
    return lettreCollision;
```

b) Concernant le parsing, nous avons rencontré plusieurs problèmes pour la sauvegarde des éléments, et l'ajout des différents éléments dans le fichier de profil. En effet, nous avons eu du mal à correctement ajouter les éléments, et après un appel de la fonction de sauvegarde le fichier était vide, sans parties sauvegardées ni même un profil. Nous avions simplement oublié de gérer nos erreurs dans le bloc catch de notre try catch, une fois que nous avons réussi à déceler l'erreur, notre fonction de sauvegarde était parfaitement fonctionnelle.

VI. Conclusion

Pour terminer, nous avons essayé à travers la réalisation de ce projet de comprendre le plus de principes de programmation liés au parsing, à la représentation et à l'organisation de données par un langage à balises, en l'occurrence XML.

Concernant le jeu, nous avons essayé de simuler une difficulté croissante à travers nos 6 niveaux, pour que chaque utilisateur ait un sentiment de progression au fur et à mesure des niveaux. Les fonctionnalités demandées pour le rendu final sont toutes implémentées à l'exception de l'ajout de mots au dictionnaire (manque de temps), et le jeu fonctionne normalement. Nous avons quand même voulu rendre l'expérience de jeu plus attractive, en ajoutant des améliorations.

Le jeu pourrait encore être amélioré, en ajoutant des changements de caméra pour Tux, des animations supplémentaires, une disparition de Tux pendant un moment lorsque l'on se trompe de lettre dans un mot. Ce jeu permet un grand nombre d'élargissements possibles !