Plan de tests

Synthèse vocale de numéros téléphoniques

Equipe:

CUMMINGS Thibaud
GOLETTO Michael
SOLEYMANKHANI Hossein
DIALLO Négué

Table de Révision

Révision	Date	Auteur	Modifications apportées
1	5 Avril 2010	groupe	Version initiale
2	29 Avril 2010	groupe	Version finale

Table des matières

1 INTRODUCTION

Ce document présente les tests effectués pour la vérification du bon fonctionnement de la librairie.

1.1 Objectifs et méthodes

Il est demandé de livrer une librairie codé en langage C permettant la synthèse vocale de numéros téléphoniques ainsi qu'une application qui teste cette librairie. La librairie qu'il est demandé de fournir doit comporter une fonction permettant la reconstruction vocale d'un numéro téléphonique. Afin de restituer vocalement des numéros de téléphones, il est nécessaire d'implémenter un algorithme de synthèse vocale. A partir d'une série de nombres en entrée, l'algorithme devra être capable de générer le fichier son correspondant à ces nombres par une voix reconstituée.

1.2 Documents de référence

Le deux documents de référence ici sont le cahier des charges (*S1*), le cahier de recette (*S2*), le plan de développement (*S3*) et la documentation interne du code (*R2*)

2 ORGANISATION

Ayant adopté un cycle de programmation en V, les phases de tests dans l'ordre (du plus spécifique au plus global) sont les suivantes:

- **tests unitaires**: les plus détaillés et spécifiques; ils s'assurent du bon fonctionnement de chaque modules individuellement.
- **tests d'intégration**: séries de tests s'assurant de la bonne communication entre chaque modules.
- **tests des recettes**: les plus globaux, une série de tests démontrant le bon fonctionnant de la librairie finale.

2.1 Tests des recettes

Les tests des recettes sont listés dans le document "cahier de recette" ref. *S2* dans le partie 3. Vérification. Veuillez vous-y référer pour les détails.

2.2 Tests d'intégration

Les tests d'intégration consistent à s'assurer que tous les modules fonctionnent les uns avec les autres. Les tests sont pas listés, il est seulement expliqué comment ils ont étés effectués.

2.3 Tests unitaires

Les tests unitaires consistent à tester chaque module individuellement.
Chaque test est expliqué.

3 TESTS D'INTÉGRATION

Les tests d'intégration ont étés effectué au fil du temps. Dès qu'un module était terminé et testé; on intégrait celui-ci. Ainsi on intégrait chaque module les uns à la suite des autres en effectuant des tests s'assurant que chaque module communique correctement avec tous les autres.

4 TESTS UNITAIRES

4.1 Interface utilisateur

Le module *interface utilisateur* permet la saisie utilisateurs des données utiles à la fonction principale. Les entrées sont :

- une chaine de caractères : numéro à synthétiser.
- un entier (0 ou 1) : voix féminine ou masculine
- un entier (0, 1 ou 2) : vitesse de lecture normale, lente ou rapide

Le s'assure que les entrées sont bien respectées:

- chaine entrée ne doit pas être nulle c'est-à-dire contenir aucun caractères
- l'entier correspondant à la voix ne peut qu'être égal à 0 ou 1
- l'entier correspondant à la vitesse ne peut qu'être égal à 0, 1 ou 2

Si les valeurs entrées ne sont pas respectées, le module les redemandera jusqu'à ce qu'il obtienne une entrée correcte.

Ce module étant très basique, il a été testé "manuellement". Nous avons entré des valeurs différentes de 0 et 1 (ou 2 pour la vitesse) pour les entrées *voix* et *vitesse* et vérifié que le module nous redemandais bien d'entrer une valeur correcte. De même pour la numéro entrée, nous avons essayé de n'entrer aucun caractère et vérifié que le module redemandais le numéro.

4.2 Traitement de la chaîne

Le module *traitement de la chaîne* prend en entrée une chaîne de caractères (numéro entré par l'utilisateur) envoyé par le module "interface utilisateur". Le module renvoi le tableau de chaînes de caractères correspondant au numéro entrée nettoyé (suppression caractères non-acceptés) et segmenté (par paires ou en fonction des séparateurs).

Le programme de teste de ce module génère une chaîne de caractères (*ASCII*) aléatoire qu'il "envoi" au module *traitement de la chaîne*. Par la suite nous avons observé le tableau retourné par le module en s'assurant que les caractères non acceptés étaient bien supprimés et que la chaine était bien segmentée (soit par paire soit en fonction des séparateurs).
Même si celui-ci fonctionnait, le programme de test n'étant pas optimal, on se rendait compte qu'il ne générait que peu de chiffres dans la chaine.
C'est pour cela que nous avons poursuivi les tests "manuellement"; c'est-à-dire en entrant nous mêmes des chaînes. Cette fois-ci nous nous sommes assurés que les chaînes entrées ressemblait plus à des numéros téléphoniques.

4.3 Analyse syntaxique / Chargements chemins enregistrements

La module *analyse syntaxique* prend en entrée un tableau de chaînes de caractères [maximum 4]; chaque chaine étant un numéro composé des caractères ascii '0' à '9'. En sortie le module renvoi le tableau de chaîne caractères correspondant au chemin vers le fichier correspondant aux enregistrements qui composent chaque numéro du tableau en entrée. (de la forme "sons/masculin/cent.wav")

Le programme de test de ce module génère des tableaux de chaînes de caractères (composés exclusivement des caractères ASCII de 0 à 9) identiques à ceux pris en entrée par le module.

Les combinaisons possibles pour chaque case sont:

- pour 1 caractère : 0 à 9 soit 10 possibilités;
- pour 2 caractères : 00 à 99 soit 10^2 possibilités;
- pour 3 caractères : 000 à 999 soit 10^3 possibilités;
- pour 4 caractères : 0000 à 9999 soit 10^4 possibilités.

Pour chaque case du tableau il y a donc $10 + 10^2 + 10^3 + 10^4 = 11110$ combinaisons possibles.

La séquence des cases n'a pas d'influence sur le résultat; c'est pour cela que l'application de test génère un tableau dont chaque case est composée de ces 11110 combinaisons (dans l'ordre de 0 à 9999).
Ce tableau est envoyé au module analyse syntaxique qui s'exécute.
Par la suite, le programme de test écrit, dans un fichier; le contenu de chaque case du tableau que génère le module.
Le fichier des résultats obtenus est écrit/formaté de la manière suivante :

```
T[0] = 0  =>  sons/feminin/zero.wav  |  sons/feminin/vide.wav
T[9] = 9  =>  sons/feminin/neuf.wav  |  sons/feminin/vide.wav

                                    [...]

T[31] = 21  =>  sons/feminin/vingtt.wav  |  sons/feminin/et.wav  |
sons/feminin/un.wav  |  sons/feminin/vide.wav

                                    [...]

T[927] = 817  =>  sons/feminin/hui.wav  |  sons/feminin/cent.wav  |
sons/feminin/di.wav  |  sons/feminin/sept.wav  |  sons/feminin/vide.wav

                                    [...]

T[1115] = 0005  =>  sons/feminin/zero.wav  |  sons/feminin/zero.wav  |
sons/feminin/zero.wav  |  sons/feminin/cinq.wav  |  sons/feminin/vide.wav

                                    [...]

T[11109] = 9999  =>  sons/masculin/neuf.wav  |  sons/masculin/mille.wav  |
sons/masculin/neuf.wav  |  sons/masculin/cent.wav  |
sons/masculin/quatree.wav  |  sons/masculin/vingt.wav  |
sons/masculin/diz.wav  |  sons/masculin/neuf.wav  |  sons/masculin/vide.wav
```

Si le module était plus complexe, nous aurions du envisagé des vérifications automatisés.

Le module est simple algorithmiquement donc la vérification des résultats du test s'est faite "manuellement" c'est à dire en vérifiant chaque ligne du fichier.
Lors de la vérification nous devions prêter plus d'attention aux résultats particuliers (les moins évidents à coder) comme par exemple : 8826 => "hui-mille-hui-cent-vingtt-six" ou 71 => "soixante-et-onze".

4.4 Lire/écrire .wav

Le module de lecture prend en entrée un fichier audio .wav (16-bit mono 44,1kHz) et renvoi une structure spécifique contenant toutes les champs d'informations d'un fichier .wav.
Le module d'écriture prend en entrée un tableau d'échantillons et renvoi un fichier .wav (16-bit mono 44,1kHz) contenant exactement les échantillons passés en entrées.
Il n'était pas intuitif de tester ces deux modules séparément; c'est pour cela qu'ils ont été testé conjointement. Le module écrire prend directement les données générés par le module de lecture.

Le test de ces deux modules consiste à faire passer un fichier audio .wav (16-bit mono 44,1kHz) en entrée puis écouter le fichier .wav généré et comparer le résultat avec la lecture de ce même fichier par un lecteur audio multimédia.

4.5 Concaténation

Le module *concaténation* prend en entrée une série ordonnée de fichiers audio .wav (16-bit mono 44,1kHz) et les concatènes les uns à la suite des autres pour ne former qu'un seul fichier .wav.
Pour une transition fluide entre chaque fichiers, un "crossfade + overlap" est utilisé: là fin de chaque fichiers chevauche le début du suivant. (cf. documentation de la fonction de concaténation)

Le test de ce module consiste à 'envoyer' plusieurs fichiers .wav au module et d'écouter le fichier wav généré. L'écoute se concentrera sur la fluidité du rendu.

4.6 Modification vitesse

Le module *modification vitesse* prend en entrée un réel (vitesse désirée: 0.7 pour plus rapide; 1.3 pour plus lent) et un fichier audio .wav (16-bit mono 44,1kHz) et renvoi un ce fichier .wav sur lequel est appliqué modification de vitesse sans affecter le ton.

Le test de ce module consiste à écouter le fichier .wav généré et s'assurer que l'augmentation ou la diminution de vitesse sans altérer le ton s'effectue et est perceptible.

4.7 Jouer son

Le module *jouer son* prend en entrée un fichier audio .wav (16-bit mono 44,1kHz) et lit celui-ci.

Pour tester cette fonction nous avons mis en paramètres d'entrée des fichiers audio .wav (16-bit mono 44,1kHz) et comparé le résultat avec la lecture de ces mêmes fichiers par un lecteur audio multimédia.

5 GLOSSAIRE

-A-
ASCII : norme de codage de caractères en informatique.

-S-
SDL_Audio : librairie libre permettant de jouer des fichiers audio.

-I-
intégration : phase de mise en relation des modules d'un programme informatique

-W-
WSOLA : algorithme *Wave-form Similarity-based Over-Lap Add* est une technique de modification de vitesse sans altérer la fréquence.

WAVE : format standard pour stocker de l'audio numérique

6 RÉFÉRENCES

Cahier des charges *S1*

Cahier de recette *S2*

Plan de développement *S3*

Documentation interne du code *R2*

7 INDEX