Licence SPI 2ème année
Algorithmique Avancée et Outils de programmation
2015-2016
Claudine Piau-Toffolon

Dans le cadre de l'UE *Algorithmique Avancée et Outils de développement*, il s'agit de produire un programme (fichiers source et un exécutable testé et opérationnel avec la documentation), réalisé avec des outils de programmation dont le sujet est présenté dans la suite du document. Le développement de ce programme se fera obligatoirement sous environnement Linux ou Mac, en langage C.

Il s'agit de valider les compétences du cours par l'écriture d'un programme original d'environ 100 lignes en langage C, correspondant à 8 à 10 heures de travail effectives réalisée essentiellement en TP.

Présentation du jeu

Le jeu Ruzzle est un jeu sur ordiphone/smartphone qui s'inspire des jeux de société Buggle et Scrabble.
L'objectif de ce jeu est de « former le plus de mots possibles avec les seize lettres disponibles dans une grille de quatre par quatre. Les mots doivent être au moins de deux lettres, et trouvés en utilisant des lettres adjacentes [dans les huit directions] les unes aux autres sans réutiliser deux fois la même case de la grille. Les formes conjuguées des verbes sont acceptées.
Comme au Scrabble chaque lettre a une valeur en fonction de sa difficulté à être placée dans un mot et certaines cases sont spéciales :

- Lettre compte double : la lettre dans cette case compte deux fois plus de points
- Lettre compte triple : la lettre dans cette case compte trois fois plus de points
- Mot compte double : les mots contenant cette case comptent deux fois plus de points
- Mot compte triple : les mots contenant cette case comptent trois fois plus de points

(Wikipédia, 26/08/2014)

La figure 1 montre une copie d'écran du jeu Ruzzle.

Figure 1 : Le Jeu Ruzzle

Structure du programme et Types Abstraits de Données (TAD)

L'objectif de ce projet est de créer un programme qui donne la liste des mots (en ordre décroissant de leur score) qui permet de maximiser le score. Le programme devra écrire cette liste sur la sortie standard (sur les PC en salle machine avec le dictionnaire de mots disponible sur l'espace de cours-section Projet).

Ce programme, acceptera deux paramètres :
1. Une chaîne de caractères décrivant la grille. Elle sera composée de 16 fois (pour les 16 cases de la grille, par ligne puis par colonne) 4 caractères :

- un caractère pour la lettre d'une case
- un caractère pour le nombre de points associé à cette lettre
- deux caractères indiquant le bonus optionnel (deux espaces si pas de bonus)
    - LD : pour Lettre compte Double
    - LT : pour Lettre compte Triple
    - MD : pour Mot compte Double
    - MT : pour Mot compte Triple

2. Le nom d'un fichier contenant un dictionnaire sérialisé minimisant son temps de chargement.

Définissez au moins les TAD suivants :

- `Mot` qui permet de représenter un mot du dictionnaire ;
- `Dictionnaire` ;
- `Case` qui représente une case de la grille du Ruzzle ;
- `CasesContigues` qui représente une succession de cases contiguës d'une grille Ruzzle ;
- `Grille` qui représente une grille Ruzzle.

Il est à noter que les fonctions `Mot` et `Dictionnaire` pourront être utilisés dans d'autres contextes (généralisez le problème).
Donnez l'analyse descendante[1] de l'opération `resoudreRuzzle`.

Méthodologie & Consignes de travail

Vous travaillerez en groupe de 2 ou 3 personnes et suivrez une méthodologie classique de développement comprenant des phases :

- ***d'analyse*** (voir la partie Structure du programme ci-dessus),
- ***de conception préliminaire*** consistant à donner les signatures des fonctions et procédures correspondant aux opérations de l'analyse (sans oublier de spécifier les préconditions),
- ***de la conception détaillée*** consistant à proposer les algorithmes des fonctions et procédures les plus compliquées.
- ***d'implémentation*** qui devra être réalisée en utilisant le langage C. Vous devrez veillez à structurer votre programme logiquement en plusieurs fichiers et préparer tous les .h en structurant le répertoire `include` (évitez que tous le `.h` soient au même niveau) et le `Makefile` pour la compilation séparée. Pour chaque `.c` commencez par créer des fonctions vides (utilisez le `Makefile` pour compiler votre fichier `XX.c` via `make src/`XX.o). Après l'écriture de chaque fonction, testez la compilation. Vous ne passerez à la fonction suivante que lorsqu'il n'y a plus aucune erreur et aucun *warning*. Structurez votre répertoire `src` à l'image de votre répertoire `include`. Il est probable, au cours du développement, que vous ayez besoin de **débogger votre code** : vous devrez alors utiliser gdb ou lldb. A l'occasion,

[1] Procèder par raffinements successifs en partant d'un problème complexe que l'on décompose en sous-problèmes moins compliqués. On décompose alors ces problèmes en sous-problèmes de plus en plus simples, jusqu'à parvenir à des problèmes tellement élémentaires que la solution en est évidente.

vous effectuerez quelques captures d'écran d'utilisation de gdb (ou lldb), où vous expliquerez le problème rencontré et la résolution mise en oeuvre.

- ***de tests unitaires et d'intégration***. Créez les tests unitaires en utilisant les axiomes et les préconditions. Ne faîtes l'intégration que si les tests unitaires passent.

Vous devrez mettre en oeuvre les autres outils de programmation avancés qui auront été présentés dans cette partie de cours. Ainsi, vous devrez réaliser de la documentation 'développeur' en utilisant **Doxygen**. Chaque groupe devra utiliser un **dépôt SVN ou GIT** de son choix (sourceforge, github, ...). L'enseignant aura accès à chacun de ces dépôts: ils devront être utilisés de manière régulière et les messages de 'log' (pour l'historique) devront être explicites.

NB : Lorsqu'un membre de l'équipe est en charge à un niveau donné (analyse, conception, développement, test unitaire) d'un probème, il ne sera pas en charge de ce même problème à un niveau différent.

Travail à rendre

Le livrable sera une archive dont le nom sera la concaténation des noms des auteurs du projet, ordonnés suivant l'ordre croissant lexicographique, au format *CamelCase*[2] et séparés par le caractère '-'.
L'extraction de cette archive créera un répertoire portant ce même nom. Il comportera :

- un répertoire `bin` qui contiendra, après exécution du `make`, les exécutables `ruzzleSolver` et `transcoder` ;
- un répertoire `lib` qui contiendra, après exécution du `make`, des bibliothèques statiques ou dynamiques ;
- un répertoire `src` qui contiendra les sources (`.c`) du projet ;
- un répertoire `include` qui contiendra les fichiers entêtes (`.h`) du projet ;
- un répertoire `doc` qui contiendra, après exécution du `make doc`, les documentations techniques html (`doc/html`) et pdf (`doc/pdf`) du projet et votre rapport au format pdf (`doc/rapport`) ;
- un répertoire `src/tests` qui contiendra les fichiers `.c` des programmes tests unitaires (utilisation de CUnit) ;
- un répertoire `tests` qui contiendra, après exécution du `make`, les exécutables des tests unitaires ;
- un fichier `Makefile` qui permettra de lancer `make` avec aucun paramètre ou avec les paramètres `clean` et `doc`.

+ Un fichier **`lisez-moi.txt`** qui explique comment compiler votre code, les fonctions qui vous semble justes, celles pour lesquelles vous avez un doute (précisez le doute!) et celles qui ne fonctionnent pas encore.

Le **rapport** demandé présentera le travail effectué :
1. Organisation du programme : découpage en fonctions, rôle de ces fonctions, explications du programme
2. Mode d'emploi du programme
3. Bilan qualitatif du travail, difficultés rencontrées, etc.

Le projet est à déposer le **20 novembre 2015** (tout retard conduira à des pénalités) sur l'espace de cours – Section Projet – Travail A Rendre

Evaluation

Elle repose sur les éléments suivants :

- Respect des consignes

[2] Pratique qui consiste à écrire la première lettre d'un identifiant en minuscule et la première lettre de chaque mot concaténé en majuscule.

- Respect de l'énoncé et des améliorations proposées
- Organisation et qualité technique du programme: découpage en fonctions, instructions, algorithmes, efficacité, gestion des erreurs...
- Présentation du programme : indentation, commentaires et nommage
- Documentation fournie
- Utilisation pertinente des outils de programmation de compilation, débogage, documentation, versionning,....