Travail Pratique #2 : AUTOMATES ET LANGAGES
LOG2810

École Polytechnique de Montréal

Trimestre : Automne 2020

Maximiliano Falicoff 2013658
Corentin Glaus 2021624
Gabriel Rouleau 2014536

Présenté à : Aurel Josias Randolph

Remis le 6 décembre 2020

Introduction :

Ce laboratoire a pour objectif d'appliquer les notions théoriques vues en classe sur les automates et des langages sur des cas concrets. Le problème de recherche de caractères est un problème fondamentale en informatique. La notion de machines à états est le fondement de sur lequel ses solutions s'y basent (comme par exemple le regex). Notre situation est similaire à celle d'un bonhomme pendu ou un premier client (ordinateur ou humain, on le notera C1) choisit un code secret et le deuxième client (un humain, C2) essaye de deviner le code secret en moins de 15 essais. A chaque essai, C2 doit entrer un mot qui compte pour 1 essai et le programme regarde si le mot correspond au code secret, si c'est le cas, le jeux est fini et C2 gagne, sinon on a deux possibilités, le mot contient des lettres qui ne sont pas dans l'alphabet ou les longueurs des chaînes ne correspondent pas; dans ce cas C2 doit rentrer un autre mot, sans perdre un essai. Si toutes les lettres sont dans l'alphabet mais son choix ne correspond pas au code, le programme lui dit combien de lettres sont erronées et on incrémente le nombre d'essais.

Présentation de la solution :

Notre solution est composée de 4 classes distinctes, soit les classes Automate, Etat, Interface ainsi que Lexique.

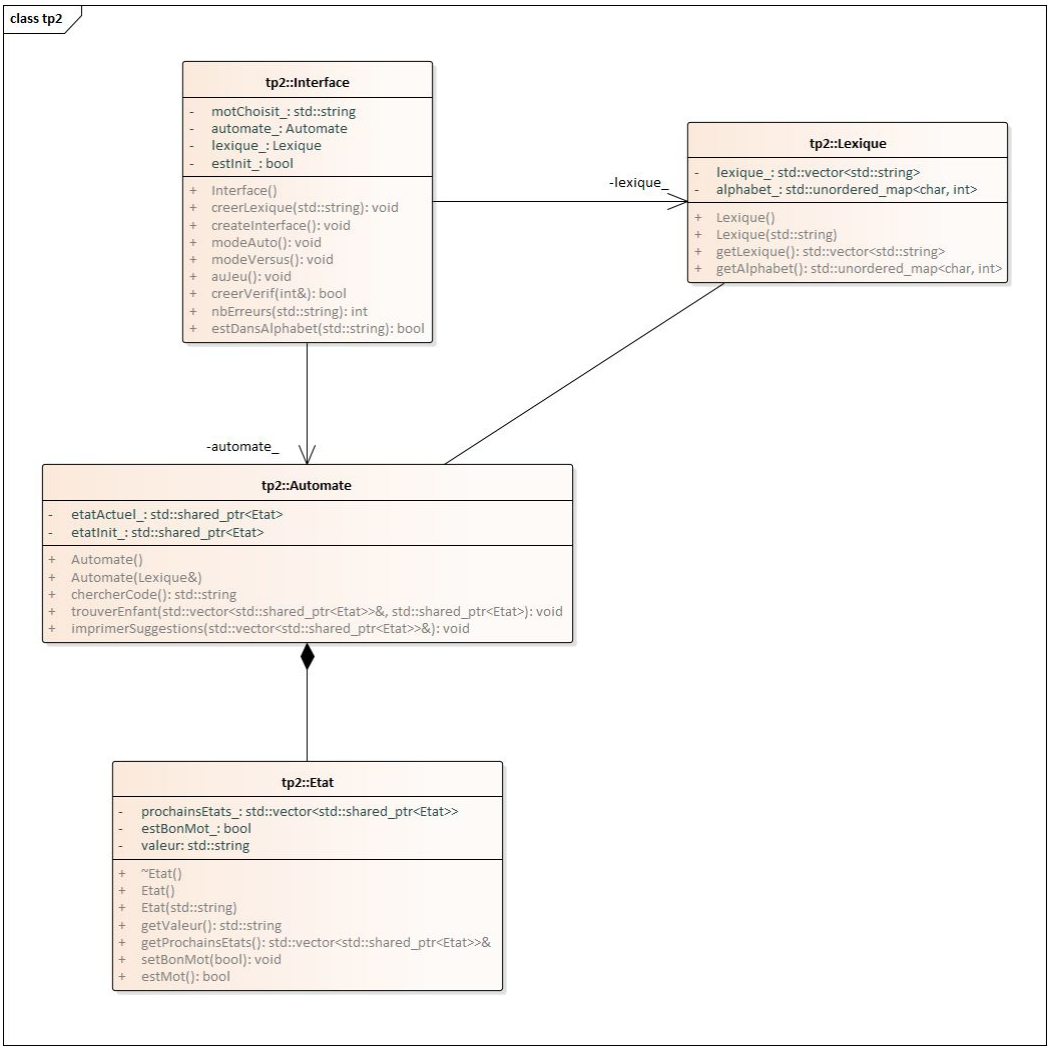
La classe Interface va être en charge de tout ce qui va être affichage d'information concernant le menu. Celle-ci va aussi être chargée de choisir aléatoirement un mot dans le lexique, de vérifier le mot entré par l'utilisateur et d'initier chacune des étapes reliées au jeu à la suite d'un signal entrée pour l'utilisateur, que ce soit un essai ou bien une direction dans le programme.

La classe automate va être chargée de lire le lexique afin d'en construire un automate. Cet automate sera par la suite utilisé par une fonction membre chercherCode et assistera le joueur 1 afin qu'il puisse choisir son mot.

La classe Etat est un constituant de la classe automate. Elle aura des informations sur si l'état est un état final, sur les prochains états ainsi que sur leur valeur courante (string correspondant au mot courant).

La classe Lexique est le conteneur de tous les mots du fichier choisi avec leur alphabet. Ainsi, c'est avec cette classe qu'il sera possible de vérifier si le mot entré par l'utilisateur est dans l'alphabet.

Diagramme de classe :

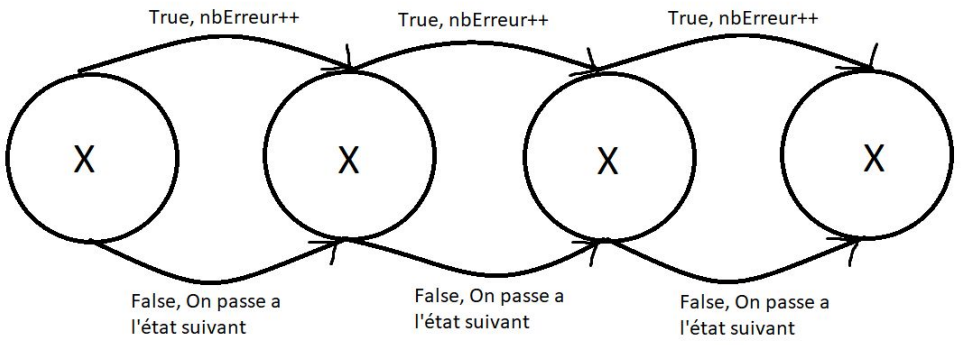


Descriptions des méthodes à implémenter:

Fonction creerLexique (C1): cette méthode se charge de créer un automate a partir d'un lexique. Dans notre implémentation, on a une classe lexique qui contient l'alphabet sous forme de map<lettre, count> ainsi qu'un vecteur contenant tous les mots du lexique. En vrai creer lexique appelle en premier le constructeur de la classe lexique avec le string du nom du fichier contenant le lexique, puis après on appelle le constructeur de l'automate en passant le nouveau lexique en paramètre (référence). On fait une boucle pour chaque mot du lexique, puis pour chaque lettre du mot on cherche si l'état existe déjà, si cela est vrai on passe au prochain état

(lettre 0 + lettre 1 exemple: mot bbbbb état 0 est b état 1 est bb). Si cet état n'existe pas, on le crée puis on peut être sûr que les prochains états pour ce mot n'existent pas. Si l'état actuel est égal au mot, ceci indique un état final.

Fonction creerVerif (C2): Cette fonction est responsable de faire la vérification de l'essai de l'utilisateur, sous la forme d'une machine de Mealy. Il est à noter que la longueur du mot ainsi que sa présence dans l'alphabet du lexique sont tout d'abord à vérifier puisqu'il faut annuler tous les essais ayant ces critères. Supposons que la vérification se rende jusqu'à celle du mot, elle sera faite de la façon d'une machine de Mealy comme illustré ci-bas. Chacune des lettres du mot écrit par le joueur seront comparées une à une avec celles du mot secret et si la comparaison (if ('char' != 'char')) est vraie, on incrémente le compteur d'erreurs. Si la comparaison est fausse, on passe à la lettre suivante (état suivant), et ce, jusqu'à ce que tout le mot soit lu. Après, le nombre d'erreurs est affiché à l'écran et si l'utilisateur a le bon mot, il est félicité.



Fonction modeAuto (C3): Cette fonction utilise simplement le lexique, qui contient la liste de tous les mots sous la forme d'un vecteur de string, et utilise rand afin de choisir aléatoirement un mot dans le lexique que l'utilisateur devra deviner par la suite. Par la suite le jeu commence en appelant la méthode auJeu().

Fonction modeVersus(C4) : Cette fonction fait l'appel à la fonction chercherCode (C1) de la classe Automate. Elle s'occupe d'aider l'utilisateur à choisir le mot à

rechercher en mode versus. Au début, la fonction parcourt tous les états jusqu'à ce qu'elle n'en trouve plus, ce qui provoque un message d'erreur et l'utilisateur doit rentrer de nouveau un code, ou alors si elle finit le parcours des états elle passe à la prochaine étape. Par la suite, la fonction regarde si l'état trouvé est un mot ou pas. Si c'est un mot, alors la fonction demande à l'utilisateur s'il veut choisir ce mot, si l'utilisateur décide oui alors la fonction se termine et retourne le code sinon la fonction redemande à l'utilisateur un nouveau mot. Si l'état n'était pas un mot alors la fonction donne des suggestions qui dérive de l'état choisi par l'utilisateur et redemande un mot.Par la suite le jeu commence en appelant la méthode auJeu().

Difficultés rencontrés :

Une des difficultés rencontrées lors du travail pratique est que le lexique n'était pas pris en mémoire dans une liste ce qui causait certains problèmes comme le choix aléatoire d'un mot qui aurait été difficile à faire dans un automate.

Une autre difficulté rencontrée est que l'automate possédait une liste de tous les états initiaux(ex: a, b, c). Ce qui a causé que certaines méthodes étaient longues et illisibles. Une solution était de créer un état vide qui possédait comme prochains états les états initiaux. Cette façon de faire simplifie l'implémentation de certaines méthodes.

Il fut aussi notablement difficile de déterminer par où commencer le projet. Nous étions incertains sur la façon de faire pour bien séparer le travail en classes distinctes ayant chacune des responsabilités dans le système. Nous avons donc procédé par modélisation UML en faisant ressortir les concepts et les objectifs du système. Ce n'est qu'après avoir mis de l'avant les requis que nous avons pu faire un diagramme de classe représentant la situation.

Conclusion :

En conclusion, ce laboratoire a été utile puisqu'il nous a aidé à mettre au clair la matière vue en classe sur les automates. Il a été aussi très utile selon nous, de mettre la théorie à la pratique dans un laboratoire comme celui-ci puisque les cours magistraux peuvent parfois être assez éloignés d'une expérience en entreprise. Il est donc certain qu'un travail comme celui-ci met de l'avant les

requis du travail professionnel, comme par exemple la communication et le travail d’équipe.