POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP2 : AUTOMATES

Session	Hiver 2019
Pondération	10 % de la note finale
Taille des équipes	3 personnes
Date de remise du projet	2 avril 2019 (23h55 au plus tard)
Directives particulières	Soumission du livrable par moodle uniquement (https ://moodle.polymtl.ca).
	Toute soumission du livrable en retard est pénalisée à raison de 10% par jour de retard.
	Programmation C++, Python ou Java
Les questions sont les bienvenues et peuvent être envoyées à: Justine Pepin (justine.pepin@polymtl.ca), Paulina Stevia Nouwou Mindom (paulina_stevia.nouwou_mindom@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++, Python ou Java.
- Notions sur les automates.
- Notions sur les structures de données.

2 Objectif

L'objectif de ce travail pratique est de vous permettre d'appliquer les notions théoriques sur les automates que nous avons vues en cours, sur des cas concrets mais hypothétiques. Dans le premier TP, votre agent jouait au jeu Qui est-ce ? Dans ce TP, il devra plutôt jouer à une version électronique simplifiée du jeu "Dont vous êtes le héros". Vous serez en charge de programmer un agent capable valider l'appartenance de langages finis à des automates et de trouver le langage généré par des automates.

3 Mise en situation

Une boîte de production de jeux vidéo a eu vent des prouesses de votre agent du premier TP et décide de vous embaucher pour développer et tester un concept expérimental de jeu à la croisée des types casse-tête et dont vous êtes le héros. Un labyrinthe de portes vous est fourni dans le dossier `Labyrinthe`. Votre agent devra parcourir le labyrinthe de la première porte `Porte1.txt` jusqu'au boss final `Boss.txt` et vaincre le boss pour gagner la partie. Le type de casse-têtes choisi consiste en un

système de mots de passe qui utilise les notions du cours de LOG2810 et tout spécialement le module de la théorie des langages. Afin de vous aider, la boîte de production vous donne un labyrinthe d'une vingtaine de portes à tester et un boss final à vaincre. La section suivante comprend des détails sur les séries de mots de passe à tester pour franchir les portes, les règles de production du langage utilisé par chaque porte, le fonctionnement de l'affrontement final avec le boss et un court rappel des notions du cours de LOG2810 sur les automates. Ainsi, vous pourrez montrer un prototype convaincant à la boîte de jeux vidéo.

4 Description

Lors de la réunion avec la boîte de production de jeux vidéo, on vous explique les détails suivants.

- Un labyrinthe est une suite de portes représentée par un dossier `Labyrinthe` comprenant plusieurs fichiers `Porte$.txt`, où $ est un numéro de porte, et un seul fichier `Boss.txt`.
- La grammaire G utilisée par toutes les portes du labyrinthe est de type $G = (V, T, S, P)$, avec l'ensemble V le vocabulaire, l'ensemble T les symboles terminaux et l'élément de départ S. Chaque porte du labyrinthe a un ensemble P de production différent de celui des autres portes.
- on considère ici que V comprend l'ensemble N des lettres majuscules de l'alphabet ainsi que les lettres minuscules $T = \{a, b, c, d, e, f\}$.
- Les règles de production des différents langages reconnus par les portes se trouvent dans les fichiers `Porte$.txt`, ainsi que les mots de passe à essayer sur ces portes. Le format d'un fichier `Porte$.txt` est détaillé dans l'annexe.
- L'agent doit toujours débuter son parcours du labyrinthe à la porte `Porte1.txt`. S'il devait se perdre dans le labyrinthe, rester coincé par une porte, car il n'identifie pas de mot de passe valide pour la débloquer, ou échouer lors de son affrontement avec le boss, il devra retourner au début du labyrinthe et débloquer la porte `Porte1.txt` à nouveau.
- Le parcours du labyrinthe par l'agent doit se faire de la façon suivante : arrivé devant une ou plusieurs portes accessibles, l'agent doit prendre connaissance d'une porte (lecture du fichier) selon son choix. Puis, il doit construire un automate pour reconnaître tous les mots du langage conformément à sa grammaire générée et pas d'autres mots. Finalement, il doit tester les mots de passe qu'il trouve dans le fichier de porte et se déplacer vers une porte indiquée par l'un des mots de passe valides.
- Il se peut qu'une porte soit un gouffre (aucun mot de passe valide ou ne conduit à aucune autre porte).
- Il est interdit que l'agent connaisse le contenu d'une porte avant qu'il ne se soit trouvé devant elle. Tant qu'elle n'est pas ouverte, une porte peut subir des modifications (c'est-à-dire qu'un labyrinthe n'est pas immuable).
- Lorsqu'une porte mène vers le boss à l'aide d'un mot de passe valide, l'agent peut enfin lire le fichier `Boss.txt`. Celui-ci contient une suite de portes qu'on pouvait emprunter pour se rendre jusqu'à lui. Si l'agent est bien passé par ce chemin, il doit créer l'automate associé à ce chemin et afficher le langage reconnu par cet automate. Il doit aussi s'assurer que la concaténation des mots de passe choisis jusqu'à présent est bien reconnue par cet automate. Si l'agent n'est pas passé par ce chemin, il perd contre le boss, doit fermer toutes les portes et doit recommencer du début son parcours du labyrinthe.

À titre d'exemple abstrait, voici un schéma possible d'un automate reconnaissant les mots 'A2A' et 'A2B' :

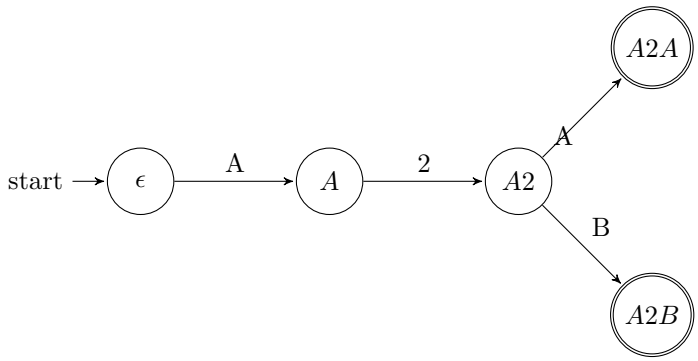

FIG. 1: Automate de validation

Un automate reconnaît un mot lorsqu'il existe une suite de transitions allant de l'état initial de l'automate à une sortie qui permet de former le mot (état final de l'automate). Ordre : la première partie du mot doit correspondre à la première transition, et ainsi de suite.

5 Composants à implémenter

C1. Écrire une fonction « ouvrirPorte() » qui lit le fichier texte contenant des règles, qui crée l'automate responsable de valider les mots de passe soumis et qui retourne une liste de portes qu'on peut ouvrir à la suite de celle passée en paramètre. Cette fonction prend en paramètre le nom du fichier à lire.

C2. Écrire une fonction « affronterBoss() » qui lit le fichier `Boss.txt`, génère l'automate associé au chemin décrit dans le fichier, valide la concaténation des mots de passe sélectionnés de l'entrée du labyrinthe jusqu'à présent et permet de trouver le langage reconnu par le boss.

C3. Écrire une fonction « genererAutomate() » qui reçoit une suite de productions dans le format de votre choix et qui crée un automate fonctionnel pouvant valider des mots de passe et fournir son langage reconnu.

C4. Écrire une fonction « afficherLeCheminParcouru() » qui indique le chemin parcouru par l'agent jusqu'à présent en précisant une suite d'événements comprenant :

   a. le choix d'une porte ;
   b. les mots de passe validés associés à cette porte avec pour chacun les portes vers lesquelles il mène ;
   c. si la porte est un gouffre et force l'agent à recommencer le labyrinthe.

   ou alors

   a. le choix du boss ;
   b. la concaténation des mots de passe depuis l'entrée du labyrinthe pour le boss et le langage universel reconnu par le boss ;
   c. si le boss est vaincu ou si le boss est vainqueur et force l'agent à recommencer le labyrinthe.

C5. Faire une interface console qui affiche le menu suivant :

   (a) Entrer dans le labyrinthe.
   (b) Ouvrir une porte.
   (c) Afficher le chemin parcouru.

(d) Quitter.

Notes

- Le programme doit toujours réafficher le menu, tant que l'option (d), ou « Quitter », n'a pas été choisie.
- L'utilisateur doit entrer un index valide, sinon le programme le signale et affiche le menu de nouveau.
- L'option (a) replace l'agent face à la première porte et réinitialise toutes les structures de données interne afin d'entamer un nouveau parcours du labyrinthe. Il garde toutefois en tête le chemin parcouru jusqu'à maintenant.
- L'option (b) permet de lire un fichier représentant une porte (ou le boss, le cas échéant) accessible à partir de l'emplacement courant, de déterminer l'automate relié à ce fichier et de déterminer les actions suivantes possibles par l'agent. Tant que l'option (a) n'a pas été choisie, l'option (b) ne peut pas être choisie. Il se peut qu'après l'exécution de l'option (b), on ne puisse plus exécuter l'option (b) parce que l'agent est tombé dans un gouffre ou a affronté le boss et qu'on doive choisir l'option (a) à nouveau.
- Après avoir sélectionné l'option (b), l'agent affiche à la console le contenu du dernier événement (Porte ou Boss) tel que demandé dans l'annexe.
- L'option (c) permet d'afficher un récapitulatif des choix effectués par l'agent en détaillant les éléments indiqués au requis C4.

Prenez note que selon votre convenance, le mot "fonction" peut être remplacé par "méthode" et vice-versa, et que plusieurs autres fonctions/méthodes peuvent être ajoutées pour vous faciliter la tâche.

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR ou tar.gz) dont le nom est formé des numéros de matricule des membres de l'équipe, séparés par un trait de soulignement (_). L'archive contiendra les fichiers suivants :

- les fichiers .cpp ou .py ou .java ;
- les fichiers .h le cas échéant ;
- le rapport au format PDF ;
- les fichiers de porte et de boss en .txt dans un dossier `Labyrinthe`, si leur format est différent de ceux fournis (vous pouvez en effet modifier le format des informations contenues dans les fichiers fournis sur Moodle, mais vous devez à ce moment-là les remettre avec vos travaux et détailler dans votre rapport le nouveau format utilisé).

L'archive ne doit pas contenir de programme exécutable, de fichier de projet ou solution de Visual Studio, de répertoire Debug ou Release, etc. Les fichiers .cpp et .h, ou .py, ou. java suffiront pour l'évaluation du travail.

6.1 Rapport

Un rapport de laboratoire rédigé avec soin est requis à la soumission (format .pdf, maximum 8 pages). Sinon, votre travail ne sera pas corrigé. Le rapport doit obligatoirement inclure les éléments ou sections suivantes :

1. Page de présentation : elle doit contenir le libellé du cours, le numéro et l'identification du TP, la date de remise, les matricules et noms des membres de l'équipe.
2. Introduction avec vos propres mots pour mettre en évidence le contexte et les objectifs du TP.
3. Présentation de vos travaux : une explication de votre solution. Ajoutez le diagramme de classes complet, contenant tous les attributs et toutes les méthodes ajoutées.
4. Difficultés rencontrées lors de l'élaboration du TP et les éventuelles solutions apportées.
5. Conclusion : expliquez en quoi ce laboratoire vous a été utile, ce que vous avez appris, etc. Vous pouvez également indiquer le temps passé sur ce TP à des fins d'ajustement pour la prochaine session.

Notez que vous ne devez pas mettre de code source dans le rapport.

6.2 Soumission du livrable

La soumission doit se faire uniquement par Moodle.

7 évaluation

éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions	2
Qualité du programme : compilation, structures de données, gestion adéquate des variables et de toute ressource (création, utilisation, libération), passage d'arguments, gestion des erreurs, documentation du code, etc.	4
Composants implémentés : respect des requis, logique de développement, etc.	
C1	4
C2	4
C3	4
C4	4
C5	3
Total de points	25

8 Documentation

- `https://www.draw.io/` (Diagrammes UML)
- `https://www.lucidchart.com/pages/examples/uml_diagram_tool` (UML)
- `learnxinyminutes.com` (Apprendre rapidement les bases d'un nouveau langage)

Annexe

1 Format des fichiers de porte

Chaque fichier `PorteN.txt` comporte entre accolades les productions pour chaque automates(une porte représente un automate). S est le symbole de départ pour toutes les portes et les autres symboles sont pris dans l'ensemble suivant : $\{A, B, C, D\}$. L'alphabet utilisé pour toute les portes est $\{a, b, c, d, e, f\}$. Le symbole "->" est une flèche et représente la relation entre les symboles du système dans une grammaire de type 3.
Après les accolades sont représentés quelques mots de passe à tester pour vérifier s'ils sont reconnus par l'automate suivis d'un espace et de la porte que ce dernier permet d'ouvrir. Par exemple "aaab Porte2" signifie le mot aaab permet d'ouvrir la porte 2. Il y a un mot de passe et sa porte correspondante par ligne.
Exemple :

```
{
S->aS, S->a
}
aa Porte3
aaba Porte15
```

2 Exemple d'affichage du chemin de l'agent

Lorsqu'on demande à l'agent d'afficher le chemin parcouru, il renvoie une liste d'événements semblable à la liste suivante :

```
Evenement Porte
a. Porte13
b. {efeddaa, Porte6, valide}, {dfefadc, Porte17, non valide}
c. Cette porte n'est pas un gouffre.
Evenement Porte
a. Porte6
b. {faaaaa, Porte9, non valide}
c. Cette porte est un gouffre, retour à Porte1.
Evenement Porte
a. Porte1
b. {cccd, Porte14, valide}, {dc, Porte7, valide}
c. Cette porte n'est pas un gouffre.
Evenement Porte
a. Porte7
b. {gg, Boss, valide}
c. Cette porte n'est pas un gouffre.
Evenement Boss
a. Porte1 Porte7
b. dcgg P={S->cA, A->cA, A->dB, S->dC, C->cB, B->, B->gB}
c. L'agent vainc le Boss.
```

Veuillez noter que l'ensemble des productions du Boss doit afficher le langage universel reconnu par toutes les portes concaténées contenues dans le Boss.

Après l'option (b), on affiche simplement le dernier événement effectué :

```
(b)
Evenement Porte
a. Porte7
b. {gg, Boss, valide}
c. Cette porte n'est pas un gouffre.
(b)
Evenement Boss
a. Porte3 Porte12 Porte9
b. dcgg P={...}
c. Le Boss vainc l'agent. Retour à la Porte1.
```