LOG2810

TP2 – Automates

Par

Gabriel Tagliabracci, 1935775

Roman Zhornytskiy, 1899786

Kevin Ciliento, 1933571

2 avril 2019

Polytechnique Montréal

Introduction

Dans le but de se familiariser avec les automates, les graphes et la génération de langage, ce TP permet de créer une version modifiée du jeu « Dont vous êtes le héros » dans lequel le participant doit choisir une série de portes afin d'arriver à battre un « boss » avec les mots de passe accumulés depuis la première porte. Un agent est créé au début du labyrinthe et celui-ci génère un automate qui lit les mots de passe des portes suivantes afin de trouver les portes valides.

Présentation des travaux

Tout d'abord, pour ce TP, nous avons décidé de coder nos fichiers en C++, ce langage nous étant le plus familier. Nous avons ensuite séparé les différentes tâches à effectuer en plusieurs classes : agent.cpp, door.cpp, automate.cpp et nodeTree.cpp.

agent.cpp

Cette classe représente l'agent s'occupant de la navigation à travers du labyrinthe et d'afficher les différentes parties du jeu. Il y a la fonction openDoors() qui prend un fichier .txt et lit le contenu. La fonction génère ensuite un automate pour valider les mots de passe des portes adjacentes. Si la porte est le « Boss », le mot de passe final est cherché et la fonction challengeBoss() est appelé par l'agent. Cette fonction vérifie que le chemin pris par l'agent est le bon et appelle ensuite la fonction concatenateAutomate() si celui-ci est bon.

La fonction concatenateAutomate() prend chacun des automates des portes choisis et les imbrique les uns après les autres. Cela veut dire qu'il prend l'état final dans le graphe de la première porte et le modifie afin qu'il devienne l'état initial de la prochaine porte. Cette fonction permet donc de créer un mot de passe pour le « Boss » à partir des mots de passe des portes du chemin choisi.

L'agent a aussi d'autres fonctions plus petites, telle la fonction clearPath() qui agit comme son nom l'indique en effaçant le trajet parcouru en le gardant toutefois encore en mémoire pour pouvoir l'afficher plus tard. Il y a aussi les fonctions d'affichage printEvent() et printBoss() qui affiche respectivement soit l'affichage standard pour l'ouverture d'une porte ou l'affichage spécialisé pour la porte du « Boss ».

door.cpp

La classe door.cpp représente les portes que l'agent doit parcourir dans le labyrinthe. Chaque porte consiste d'une série de règle de grammaire permettant de créer une série de mots de passe valides ainsi que les portes adjacentes avec leur propre mot de passe. Dans cette classe, il y a une fonction canOpen() qui vérifie si la porte est présente dans la liste des portes pouvant être ouvertes. isValid() est la méthode vérifiant la validité d'une certaine porte par la variable « validity ». Cette variable est changée dans la fonction validate() qui vérifie si le mot de passe de la porte est valide.

Cette classe a aussi des fonctions pour lire les fichiers Porte#.txt comme readFile() et readRule() qui lisent respectivement le fichier .txt et les règles de grammaires présentes dans ce même fichier. Finalement, la fonction readNextDoor() permet de trouver une porte adjacente et de l'insérer dans un map passMap_ avec son mot de passe. Tous les maps de portes adjacentes de la porte présente sont ensuite intégrés dans un multipmap doorMap_.

automate.cpp

La classe automate.cpp contient les différentes méthodes pour parcourir le graphe NodeTree. Elle possède tout d'abord la fonction generateAutomate() qui prend les règles présentes dans la porte pour créer le graphe. Les états initiaux sont insérés en tant que noyaux et, selon l'état final de la règle, ceux-ci sont insérés en tant que « child » du noyau. Cela créer donc un graphe orienté dans lequel les mots de passe traversent pour être valider. La fonction validatePassword() commence à partir du début de l'arbre et le parcoure selon la valeur des différentes règles. Si le mot de passe se termine sur un noyau terminal (qui n'a pas de « child »), alors le mot de passe est validé avec la fonction « validate() » dans Door.cpp. Il est invalide s'il se termine sur n'importe quels autres noyaux ayant des « childs ».

La fonction validatePasswords() prend tous les mots de passe des portes adjacentes et utilise la fonction validatePassword() pour trouver les mots de passe valides. Finalement, la classe peut trouver la valeur d'un « Edge » avec la méthode trouverLettre().

nodeTree.cpp

La classe nodeTree.cpp permet de créer un graphe contenant les possibilités de mots de passe valides pouvant être créer par les règles de grammaire de la porte. La classe contient deux autres structures pour sa création : Node et Edge. « Node » représente un état dans les règles (soit initial ou final) et forme un noyau dans le graphe. Ces noyaux sont ensuite reliés entre eux par des « Edge » qui prennent deux « Node » comme noyau de départ et noyau d'arrivée afin de créer une branche orientée. Ces branches contiennent une valeur qui est la lettre du mot de passe dans les règles.

Cette classe contient la fonction find() qui cherche à travers le graphe pour trouver si la paire entrée en paramètre est présente. Elle contient aussi la méthode insert() qui ajoute un nouveau « Edge » dans le vecteur contenant les « Edge » ainsi qu'un nouveau noyau s'il n'est pas déjà présent dans le graphe. Ces deux structures sont ensuite intégrées dans une paire dans le graphe.

main.cpp

Finalement, le main sert principalement à afficher l'interface du jeu donnant plusieurs options au joueur : entrer dans le labyrinthe, ouvrir une porte, afficher le chemin parcouru ou quitter. L'option « a » permet de placer l'agent à la porte 1 afin de commencer le labyrinthe. Cette option peut être sélectionnée durant le jeu ou quand le joueur arrive dans un gouffre. Le chemin déjà parcouru est gardé en mémoire même si le joueur recommence au début. L'option « b » utilise la méthode openDoor() pour ouvrir une porte sélectionnée par le joueur. Cette option ne permet d'ouvrir que les portes dont le mot de passe a été validé. L'option « c » affiche le chemin de porte parcouru par le joueur depuis le début de sa partie. Elle permet aussi d'afficher le chemin en mémoire après un recommencement de partie. L'option « d » permet de quitter le jeu.

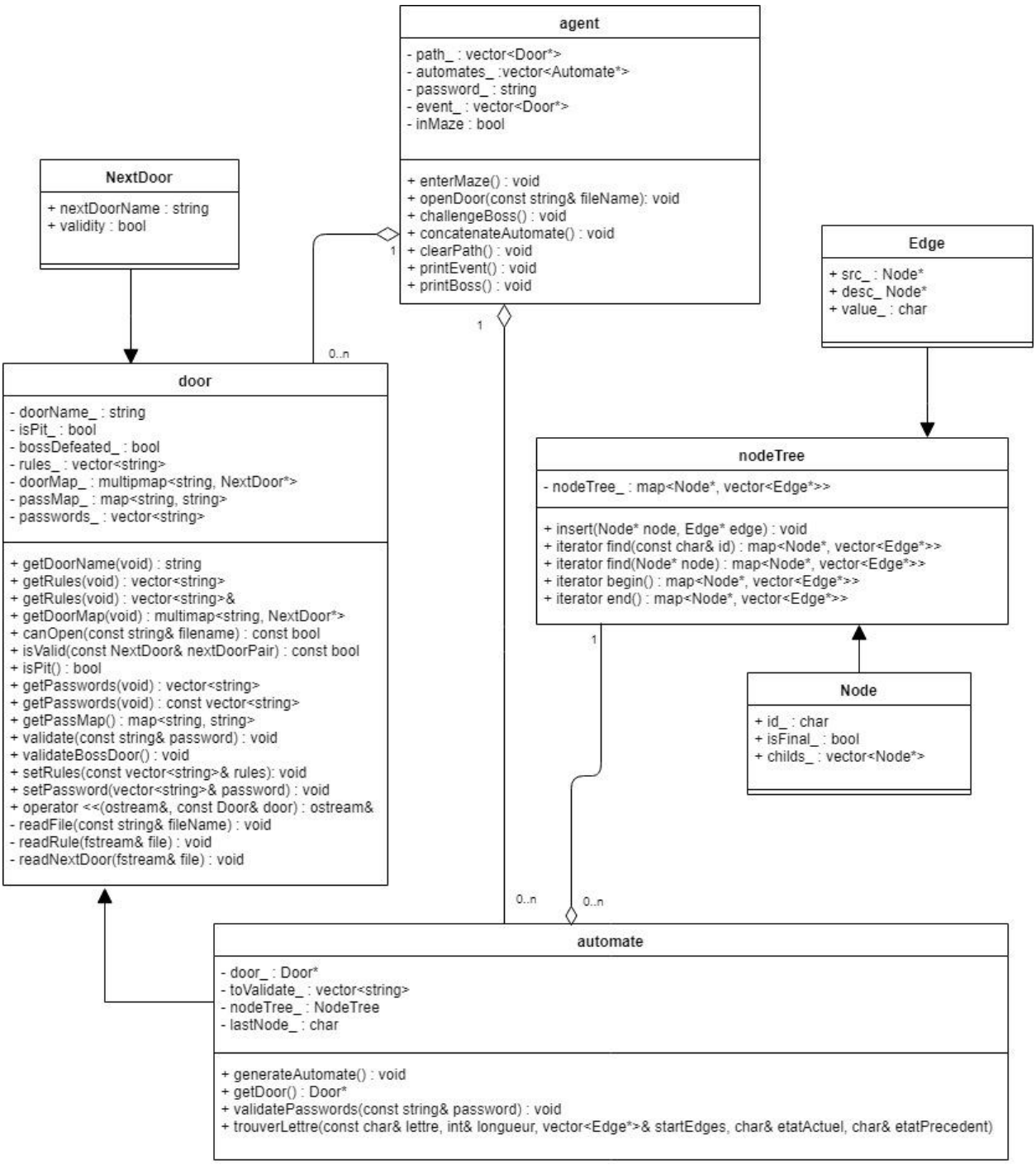

Figure 1. Diagramme de classe du logiciel.

Difficultés rencontrées lors de l'élaboration du TP

Une des difficultés que nous avons rencontré tôt dans ce travail a été la contrainte de temps. En effet, cela nous a pris beaucoup de temps à décider la façon avec laquelle nous allions procéder et coder en plus de facteurs externes au projet. Par exemple, pour créer un automate qui fonctionne, nous avons dû changer sa structure plusieurs fois afin que ses attributs ne soient pas répétitifs. Cette contrainte nous a donc empêcher d'implémenter certaines fonctions d'une manière que l'on aurait trouver encore plus optimale, mais nous avions malheureusement plus assez de temps pour ce faire.

Une autre difficulté qui est en lien avec la première est l'utilisation du polymorphisme. La porte « Boss » aurait pu être une classe héritée de la classe Door.cpp, ce qui aurait pu éviter de poser des conditions *if* dans certaines fonctions quand elle avait à distinguer entre des portes normales et la porte « Boss ». Cependant, nous avons pensé à cette idée trop tard et donc le temps restant avant la remise n'était pas suffisant pour tout changer notre implémentation. Nous sommes donc allés avec l'option consistant à intégrer des conditions *if* dans les fonctions pertinentes à la porte « Boss ».

Afficher la porte « Boss » fût aussi une autre difficulté rencontrée, car elle devait avoir une apparence particulière qui différait de l'affichage des portes du chemin. Elle devait afficher la concaténation des différents mots de passe en plus des règles de grammaires régissant ce mot de passe. Nous avons donc dû créer une fonction qui faisait l'affichage spécifique de la porte « Boss ».

Nous avons hésité au début sur la façon avec laquelle nos allions valider les mots de passe. Nous avions tout d'abord comme idée de générer tous les mots de passe possibles avec les règles de grammaire de la porte, mais nous nous sommes rapidement rendu compte que cela serait bien trop long pour le système, car le nombre de mots de passe valides pourrait être dans les centaines. Après avoir abandonné cette idée, nous avons décidé d'y aller avec un graphe que les mots de passe allaient traverser pour se valider. Nous avons ensuite rencontré une autre difficulté avec la manière de stocker les noyaux et les arcs dans le graphe. Ce problème fût résolu en les intégrant sous la forme d'un map.

Finalement, un des derniers problèmes que nous avons rencontrés a été la concaténation des automates. En effet, notre algorithme avait de la difficulté à changer l'état final du premier automate en l'état initial du deuxième automate. Cela entrainait que le mot de passe de la porte « Boss » ne pouvait se valider même si l'agent avait pris le bon chemin.

Conclusion

En conclusion, en créant un jeu « Dont vous êtes le héros » qui utilise des automates ainsi que la théorie du langage pour créer des mots de passe nous a permis de se familiariser avec le principe d'utilisation de graphe pour la validation de mots de passe ainsi que les règles de création de langage.

Au total, nous avons mis environ 24 heures pour réaliser ce travail. Nous avons mis 3-4 heures pour créer la première fonction C1 (ouvrirPorte()), 5-6 heures pour C2 (affronterBoss()), 4-5 heures pour C3 (genererAutomate()), 2-3 heures pour C4 (afficherLeCheminParcouru()), 1-2 heures pour C5 (menu), 3 heures pour la rédaction du rapport et 2 heures pour la révision du code et du rapport, ce qui fait que nous avons mis plus de temps pour ce travail que pour nos examens.