POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP1 : GRAPHES

Session	Hiver 2017
Pondération	10 % de la note finale
Lieu de réalisation	L-4708
Taille des équipes	3 étudiants
Date de remise du projet	22 février 2017 (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 $C++$ avec Visual Studio.
Les questions sont les bienvenues et peuvent être envoyées à: David Johannès (david.johannes@polymtl.ca), Mariam Tagmouti (mariam.tagmouti@polymtl.ca), John Mullins (john.mullins@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++.
- Notions de théorie des graphes.

2 Objectif

L'objectif de ce travail pratique est de vous permettre d'appliquer les notions théoriques sur les graphes que nous avons vues en cours, sur des cas concrets mais hypothétiques, tirés de votre quotidien. A cet effet, il est question dans ce travail de permettre à un explorateur d'optimiser son parcours dans un jeu du type Pokémon Go.

3 Mise en situation

Pokémon Go est un jeu en réalité augmentée, dont le but est de permettre à l'utilisateur d'attraper des Pokémons sur les trajets de son quotidien, mais aussi de visiter des arènes pour affronter des adversaires (duels de pokémons), et de visiter des stations ou pokéstop pour récolter des items intéressants comme des pokéball, de l'encens (pour attirer des pokémons), etc.

4 Description

Un étudiant du département de génie informatique et génie logiciel émet l'idée de mettre en oeuvre une nouvelle application afin d'optimiser le parcours de l'explorateur lors de sa chasse aux pokémons. Il décide d'associer à chaque objet du monde des Pokémon un gain, qui dépend bien évidemment de la valeur de l'objet et est proportionnel à celui-ci. L'étudiant a fait beaucoup de recherche sur Internet pour récolter les emplacements des divers objets (pokémons, arènes, et pokéstop), et voudrait que son application soit capable de proposer à l'explorateur un chemin optimal, par exemple pour maximiser son gain. Pour cela, il décide de représenter le monde de pokémon par un graphe des distances entre les objets de ce monde. Ainsi, le monde des Pokémons est représenté dans son application par un graphe dont les sommets correspondent aux objets, et les arcs aux distances entre deux objets. L'étudiant vous fournira bien évidemment ce graphe des distances, et il a besoin de vous pour l'aider à implémenter certains composants de son application. Lors de la séance de présentation du projet, il vous explique les concepts suivants, qui sont des concepts simplifiés du vrai jeu, mais suffisants pour faire une première ébauche de l'application. Il vous fait également un rappel sur certaines notions de théorie des graphes que vous aurez à manipuler.

- Comme dit précédemment, chaque objet du monde des Pokémons, à savoir dans notre cas les pokémons, les pokéstops, et les arènes, possède un gain proportionnel à la valeur de l'objet. Seuls trois types d'objet seront présents : les pokémons, les pokéstops, et les arènes.
- La carte du lieu dans lequel progressera l'explorateur repose sur une matrice de distances entre les différents objets présents dans ce lieu, et est donc représentable par un graphe non orienté (on suppose que l'explorateur est à pied), dont les sommets sont les objets, et les arcs les distances entre les objets. Un tel graphe sera connexe, à savoir que l'on connaîtra la distance entre un objet et tous les autres du lieu concerné.
- Les objets comme les Pokémons et les pokéstops peuvent être visités plusieurs fois dans la journée, mais ont un temps de rechargement à prendre en compte. En effet, une fois qu'un tel objet a été visité par l'explorateur, cet objet disparaîtra de la carte pendant une certaine durée de temps, ou de façon équivalente, une certaine distance à parcourir car on suppose que l'explorateur marche à la même vitesse tout au long de son trajet).
- Les Pokémons sont de trois types : les Pokémons rares, de gain 100, et qui, s'ils sont visités par l'explorateur, réapparaissent sur la carte une fois que l'explorateur aura marché 500 mètres de plus ; les Pokémons normaux, de gain 40, et qui, s'ils sont visités par l'explorateur, réapparaissent sur la carte une fois que l'explorateur aura marché 200 mètres de plus ; les Pokémons fréquents, de gain 10, et qui, s'ils sont visités par l'explorateur, réapparaissent sur la carte une fois que l'explorateur aura marché 100 mètres de plus.
- Les pokestops peuvent être également de gain variable, en fonction de leur valeur. Un pokestop réapparaît sur la carte une fois que l'explorateur aura marché 100 mètres de plus après l'avoir visité.
- Les arènes sont des objets qui possèdent également un gain (assez fort), et qui ne peuvent être visités qu'une seule fois dans la journée. Ainsi, une fois visitée, une arène doit disparaître de la carte pendant le reste du trajet de l'explorateur. Autrement dit, l'explorateur ne pourra pas visiter deux fois la même arène (mais il est possible qu'il puisse visiter deux fois ou plus les autres objets durant son trajet).
- Les trois types d'objet du monde des Pokémons sont ainsi susceptibles de disparaître un fois visités, pour réapparaître plus tard. Il ne faudra donc pas oublier de retirer les sommets correspondants dans le graphe, et de les rajouter plus tard.
- Les sommets du graphe représentant la carte devront contenir un paramètre gain. Sur un sommet du graphe donné, ce gain indique en quelque sorte la valeur de l'objet situé sur ce sommet. Le gain représente ici la propriété de notre sommet.

- Un sous-graphe est un graphe contenu dans un autre. Plus formellement, H est un sous-graphe d'un graphe G si c'est un graphe, si l'ensemble des sommets de H est un sous-ensemble de l'ensemble des sommets de G, et si l'ensemble des arêtes (ou arcs) de H est un sous-ensemble de l'ensemble des arêtes (ou arcs) de G.
- Un chemin reliant un sommet a à un sommet b est l'ensemble des arêtes consécutives reliant a à b. La séquence des arêtes parcourues définit le chemin entre a et b.
- Un graphe est connexe s'il existe toujours un chemin entre chaque paire de sommets distincts du graphe. Dans le cas contraire, un graphe est constitué de l'union de plusieurs sous-graphes disjoints, lesquels représentent les composantes connexes du graphe.
- Un graphe valué (ou pondéré) est un graphe dans lequel chacun des arcs présente une valeur. Cette valeur peut symboliser une distance, un coût, une durée, etc. Dans notre cas, les arcs représentent la distance de la route qui sépare deux objets du monde de Pokémon.
- La longueur d'un chemin, dans un graphe pondéré, est la somme des poids des arêtes parcourues.

5 Composants à implémenter

C1. Écrire une fonction récursive "creerGraphe()" qui permet de créer le graphe représentant les routes et les objets (sommets) à partir d'un fichier, dont le nom est passé en paramètre.

C2. Écrire une fonction "lireGraphe()" qui permet d'afficher le graphe (cf. annexe a. pour un exemple d'affichage de la carte sous forme de graphe).

C3. Écrire la fonction "plusCourtChemin()" qui permet de déterminer, en vous inspirant de l'algorithme de Dijkstra, le plus court chemin pour atteindre un gain minimal passé en paramètre. L'origine (point de départ) et le gain voulu doivent être passés en paramètres. La fonction affiche le gain final obtenu (qui doit être au moins égal au gain passé en paramètre), le plus court chemin utilisé et la longueur de ce dernier.

C4. Écrire la fonction"plusGrandGain()" qui permet de déterminer, en vous inspirant de l'algorithme de Dijkstra, le plus grand gain que l'explorateur peut obtenir en parcourant au maximum une distance passée en paramètre. L'origine (point de départ) et la distance maximale à parcourir doivent être passées en paramètres. La fonction affiche le gain final obtenu, le plus court chemin utilisé et la longueur de ce dernier (qui ne doit pas dépasser la distance maximale passée en paramètre).

C5. Faire une interface qui affiche le menu suivant :

   (a) Mettre à jour la carte.
   (b) Déterminer le plus court chemin (et demander le gain minimal à obtenir)
   (c) Déterminer le plus grand gain (et demander la distance maximale à parcourir)
   (d) Quitter

Notes

- Le programme doit toujours réafficher le menu, tant que l'option (d) n'a pas été choisi.
- L'utilisateur doit entrer un index valide, sinon le programme le signale et affiche le menu de nouveau.
- L'option (a) permet de lire une nouvelle carte afin de créer le graphe correspondant. Pour ce faire, le nom du fichier contenant les informations de la carte doit être demandé. Il est demandé d'afficher le graphe obtenu à la lecture d'un fichier. Le format du fichier est décrit en annexe.

- L'option (b) permet de déterminer le plus court chemin en sachant le point de départ et le gain minimal à atteindre. Pour ce faire, les informations sur l'origine (point de départ) et le gain minimal voulu doivent être demandées.
- L'option (c) permet de déterminer le plus grand gain en sachant le point de départ et la distance maximale à parcourir. Pour ce faire, les informations sur l'origine (point de départ) et la distance maximale que l'explorateur veut parcourir doivent être demandées.
- Si une nouvelle carte est lue, les options (b) et (c) doivent être réinitialisées.

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. La figure 1 présente un exemple de diagramme de classes à partir duquel vous pouvez travailler.

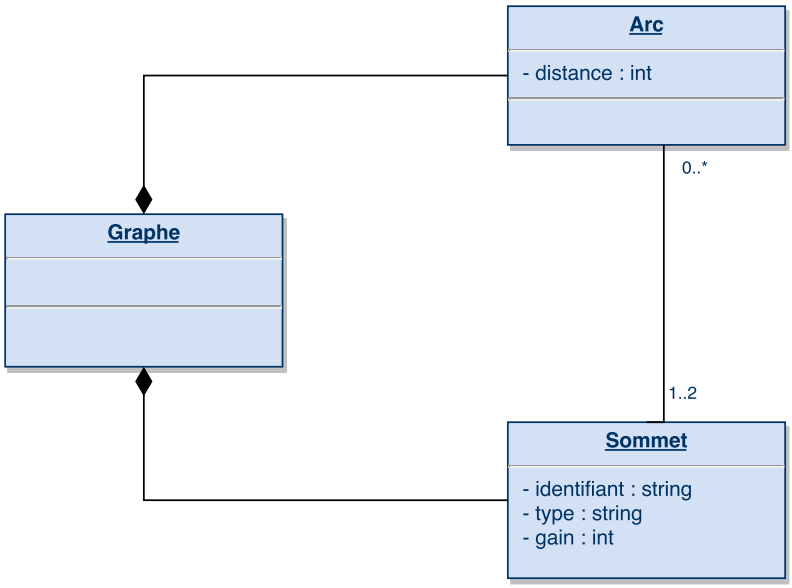

Figure 1: Exemple du diagramme de classes de la solution

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR) 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;
- les fichiers .h le cas échéant;
- le rapport au format PDF;
- le fichier .txt passé en argument (option (a)), s'il est différent de celui fourni par l'instructeur du laboratoire (vous pouvez en effet modifier le format des informations contenues dans le fichier fourni sur Moodle, mais vous devez à ce moment-là le remettre avec vos travaux).

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 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 (aussi bien le code source que l'exécutable). Le rapport doit obligatoirement inclure les éléments ou sections suivantes :

1. Page 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. Vous pouvez compléter la page présentation qui vous est fournie.
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, vos attentes par rapport au prochain laboratoire, etc.

Notez que vous ne devez pas mettre le 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.	2
Composants implémentés : respect des requis, logique de développement, etc.	
C1	3
C2	3
C3	3
C4	3
C5	4
Total de points	20

8 Documentation.txt

- http://www.cplusplus.com/doc/tutorial/
- http://public.enst-bretagne.fr/∼brunet/tutcpp/Tutoriel%20de%20C++.pdf
- http://fr.openclassrooms.com/informatique/cours/programmez-avec-le-langage-c
- Algorithme de Dijkstra illustré : https://www.youtube.com/watch?v=0nVYi3o161A

Annexe

1 Plus court chemin

Soient un graphe valué (ou pondéré) G et deux sommets a et b de G. Pour calculer le plus court chemin entre a et b, utilisons l'algorithme de Dijkstra. Soit *d(x)*, la distance du sommet x par rapport au sommet de départ a, *w(u,v)*, la longueur d'une arête $\{u,v\}$ et *ch(a,x)*, le chemin (liste des arêtes traversées) du sommet a au sommet x.

- Au début de l'algorithme, les distances de chaque sommet x au sommet de départ a sont fixées à la valeur infinie ($d(x) = \infty$), à l'exception du sommet de départ, a, dont la distance (par rapport à lui-même) est 0. Pour chaque sommet, on associe également la liste des sommets traversés (le chemin emprunté) du sommet initial a jusqu'au sommet en question. À cette étape de l'algorithme, cette liste est vide pour tous les sommets, sauf pour le sommet a, dont la liste se résume à lui-même. En d'autres termes, pour tous les autres sommets x de G, on associe une étiquette "x, ∞, ()", et pour le sommet a, on a "a, 0, (a)". On considère également un sous-graphe vide S.
- À chaque itération, on sélectionne le sommet x de G, qui n'apparaît pas dans S, de distance minimale (par rapport au sommet a). Ce sommet x est ajouté au sous-graphe S. Par la suite, si nécessaire, l'algorithme met à jour les étiquettes des voisins x_v du sommet x ajouté. Cette mise à jour s'effectue si $d(x_v) > d(x) + w(x,x_v)$. Dans ce cas, l'étiquette du sommet x_v est modifiée comme suit:
  $\{x_v, d(x) + w(x,x_v), (ch(a,x),x_v)\}$.
- On répète l'opération précédente jusqu'à la sélection du sommet d'arrivée b, ou jusqu'à épuisement des sommets. L'étiquette associée à b donne alors le plus court chemin de a à b, de même que la longueur de ce dernier.

2 Informations utiles

(a) **Affichage de la carte**
Pour afficher la carte, il faut indiquer, pour chaque sommet, quel est son type (pokemon, pokestop, arene), et la liste des sommets voisins avec les distances associées, comme illustré ci-dessous (par exemple) :
(objet1, typeObjet1, gainObjet1, ((objet_voisin$_{1_1}$, distance$_{1_1}$), (objet_voisin$_{2_1}$, distance$_{2_1}$), ..., (objet_voisin$_{n_1}$, distance$_{n_1}$)))
(objet2, typeObjet2, gainObjet2, ((objet_voisin$_{1_2}$, distance$_{1_2}$), (objet_voisin$_{2_2}$, distance$_{2_2}$), ..., (objet_voisin$_{n_2}$, distance$_{n_2}$)))
...

(b) **Affichage du parcours**
Pour afficher le plus court chemin, la liste des sommets (ou objets) définissant le trajet doit être présentée comme suit :
$point_{départ} \rightarrow point_1 \rightarrow point_2 \rightarrow ... \rightarrow point_n \rightarrow point_{arrivée}$

(c) **Structure des fichiers**
Les cartes sont présentées sous forme de fichier textes (cf. le fichier texte fourni sur Moodle). Un tel fichier contient deux listes :

- La première ligne contient la liste des sommets. Chaque sommet est constitué d'un triplet (*identifiant*, *type*, *gain*). Le *type* est soit *pokemon*, soit *pokestop*, soit *arene*. Le *gain* est un entier.
- La seconde ligne contient la liste des distances entre chaque paire de sommets, séparées par des point-virgules. Chaque distance/route peut être empruntée dans les deux sens. Chaque distance précise ainsi deux sommets et une distance séparant ces deux sommets (en mètres).