|  |  |
| --- | --- |
| LogoPoly_Genie1Classe.png | École Polytechnique de Montréal  Département de Génie Informatique et Génie Logiciel |

**LOG2810**

**STRUCTURES DISCRETES**

**Hiver 2017**

**TP1 : Graphes**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Remis par :**

|  |  |
| --- | --- |
| **Matricule** | **Prénom & Nom** |
| **1801383** | **Philippe Courtemanche** |
| **1718526** | **Anthony Dentinger** |
| **1769769** | **Marc-Gaël Hounto** |

**À :**

**<David Johannès>**

**Le 1er mars 2017**

# Introduction

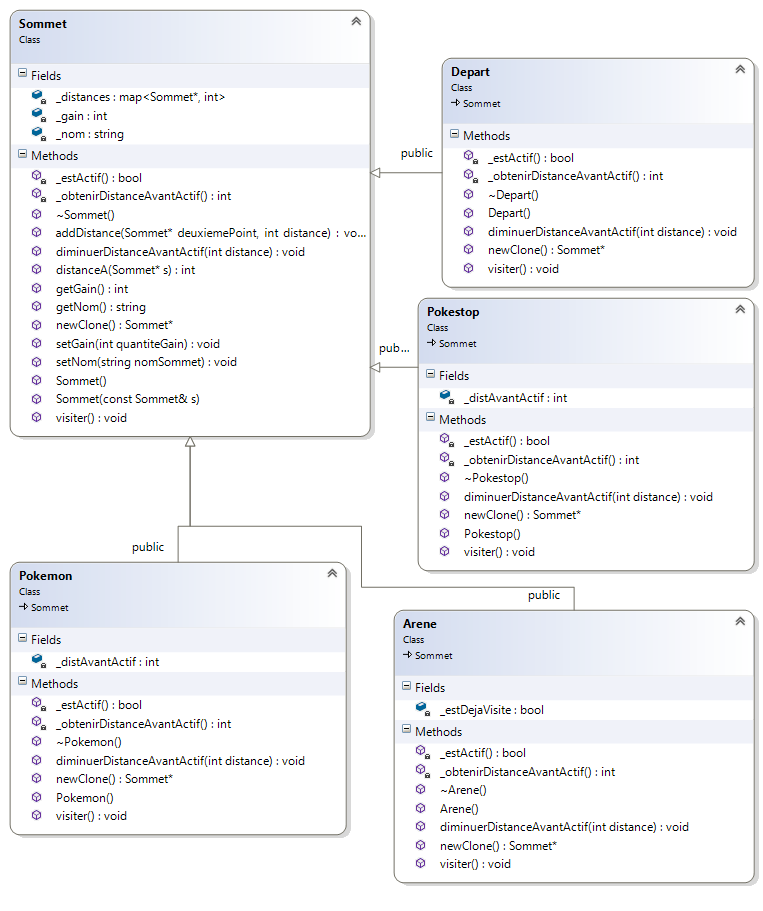
Le contexte de ce travail pratique est celui où un étudiant de génie informatique et logiciel nous demande de l’aide pour développer un exécutable qui permet d’optimiser les parcours de joueurs de Pokémon Go. Pokémon Go est un jeu dont le but est d’attraper des pokémons, de visiter des arènes pour combattre des adversaires et de visiter des pokéstops pour trouver des items intéressants. Puisque c’est un jeu en réalité augmentée, le joueur doit se déplacer dans le monde réel afin de se rendre à ces endroits. Malheureusement, ces déplacements sont couteux en termes de temps et d’énergie pour les joueurs et c’est pourquoi cet étudiant propose de développer un exécutable permettant d’optimiser le parcours des joueurs.

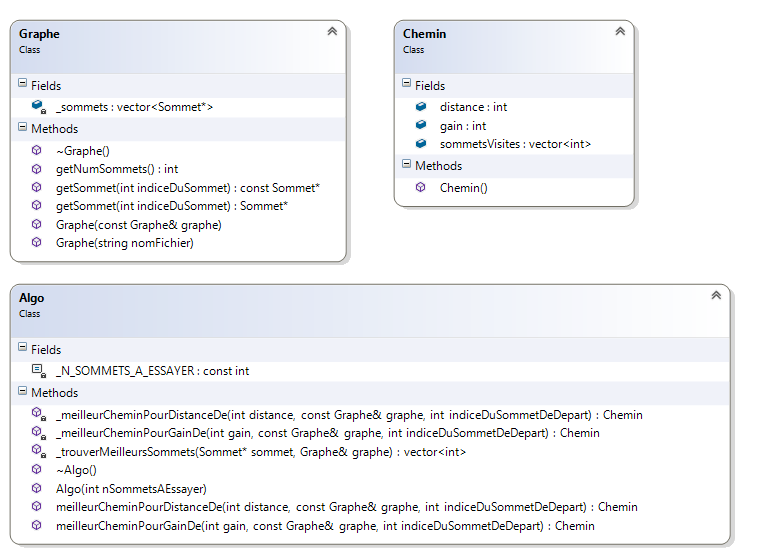
L’objectif est donc d’appliquer les notions théoriques sur les graphes, afin d’optimiser le parcours des joueurs de Pokémon Go. Il sera donc nécessaire de créer et afficher un Graphe à partir d’un fichier texte contenant les emplacements, les gains associés aux sommets, ainsi que les distances entre tous les sommets. En effet, puisque nous appliquons la théorie associée aux graphes, les pokémons, arènes et pokéstops, que nous appellerons désormais « sommets », auront des gains associés à eux, ainsi que des distances entre eux. De plus, deux fonctions d’optimisation devront être implémentées : la fonction « plusCourtChemin() » et la fonction « plusGrandGain() ». La fonction « plusCourtChemin() » permettra d’obtenir le chemin le plus court permettant d’obtenir un gain minimal entré en paramètre. Inversement, la fonction « plusGrandGain() » permettra d’obtenir le chemin comportant le plus grand gain tout en parcourant une distance plus petite que la distance maximale entrée en paramètre. Enfin, une interface permettra de mettre à jour la carte, déterminer le plus court chemin, déterminer le plus grand gain et quitter.

# Présentation du travail

Premièrement, la classe Graphe lit le fichier texte contenant les informations par rapport aux sommets. Un objet de la classe Graphe est donc construit. Ensuite, les algorithmes « meilleurCheminPourDistanceDe() » et « meilleurCheminPourGainDe() » utilisent une méthode qui est celle d’essayer les deux meilleurs chemins, ceux-ci sont déterminés à partir de leur gain par mètre. Lorsque l’algorithme découvre une portion du chemin qui comporte une boucle, il répète cette boucle jusqu’à ce qu’à obtenir la valeur cible. D’autre part, afin de tenir compte du temps de rechargement, lorsqu’un sommet est visité ce sommet est noté comme étant plus distant des autres points qu’il n’est physiquement. Enfin, l’information est relayée à l’interface graphique pour que l’utilisateur voit le résultat. L’utilisateur peut aussi mettre à jour la carte et quitter à partir de l’interface.

## Diagramme de classes :





# Difficultés rencontrées

Nous nous sommes rendus compte que l’algorithme de Djikstra ne calculait pas le meilleur chemin, donc nous avons trouvé la solution de visiter les quelques meilleurs chemins tel que décrit dans la section précédente.

D’autre part, nous avons éprouvé de la difficulté à lire le fichier texte en le stockant tout d’un coup dans une variable de type string. Éventuellement, un de nous a suggéré que nous utilisions un « string stream » : dans une première boucle, sauvegarder ce qui était entre « ; » dans un string et ensuite le convertir en « string stream » pour isoler les éléments entre virgules.

Nous avons aussi rencontré des difficultés de performance, puisque l’algorithme prenait plusieurs minutes à s’exécuter pour une distance supérieure à mille mètres. L’utilisation des boucles tel que décrit précédemment a permis de solutionner ce problème.

D’autre part, étant donné que nous n’avons pas pu compiler notre programme avant qu’il ne soit entièrement terminé, nous avons passé un certain temps à corriger les bogues.

# Conclusion

Ce laboratoire nous a permis d’appliquer les notions sur les graphes que nous avons vu en classe telles que la notion de sommet, d’arc, de graphe connexe et de chemin. Il nous a aussi permis de concevoir un système en équipe tout en appliquant des nouvelles notions de mathématiques, une compétence que nous n’avons pas souvent l’occasion de pratiquer, mais qui sera très utile dans le futur. D’autre part, nous avons appris à modéliser un graphe en utilisant des classes.

Nous avons pu revoir les notions de la programmation orientée objet en C++, ainsi que celle de la lecture de fichier. Nous avons appris à réfléchir sur papier en concevant nos classes et nos algorithmes avant de passer à l’écriture du code.

Pour ce qui est du prochain laboratoire, nous nous attendons à devoir résoudre un problème en nous servant de notions de mathématiques discrètes vues en classe et d’un langage de programmation orienté objet tel que le C++. Puisque le prochain laboratoire est à remettre environ au même moment que le cours douze, nous pensons que cette notion de mathématique discrète sera peut-être celle du dénombrement, de la théorie des langages ou des automates.