|  |  |
| --- | --- |
| 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** |
| **1798345** | **Adam Martin-Côté** |
|  | **Jonathan Gervais-Ranger** |
|  | **Louis Cormier** |

**À :**

**David Johannès,**

**Mariam Tagmouti**

**Le 2 mars 2017**

Table des matières

1. [Introduction 3](#_Toc476076968)
2. [Présentation de la solution 3](#_Toc476076969)

[Diagramme de classes 3](#_Toc476076970)

[Implémentation 3](#_Toc476076971)

[Fonction créerGraphe 3](#_Toc476076972)

[Fonction lireGraphe 3](#_Toc476076973)

[Fonction plusCourtChemin 3](#_Toc476076974)

[Fonction plusGrandGain 3](#_Toc476076975)

1. [Difficultés rencontrées 3](#_Toc476076976)

[Fonction créerGraphe 3](#_Toc476076977)

[Fonction lireGraphe 3](#_Toc476076978)

[Fonction plusCourtChemin 3](#_Toc476076979)

[Fonction plusGrandGain 3](#_Toc476076980)

1. [Conclusion 3](#_Toc476076981)

# Introduction

Le concept des graphes est étudié depuis plusieurs années. Plusieurs grands mathématiciens ont beaucoup travaillé avec, notamment Dijkstra, Euler et Hamilton. De nos jours, il est utilisé dans plusieurs domaines, dont l’informatique, puisqu’il permet de faire une très bonne structure de données.

De ce fait, il nous a été demandé d’utiliser les connaissances acquises durant le cours de *Structures discrètes* sur les graphes afin d’aider les joueurs de Pokémon Go à optimiser leurs déplacements. Nous avions donc à créer une application qui permettrait aux joueurs de savoir qu’elle est le plus court chemin afin d’obtenir un gain désiré et aussi de savoir qu’elle est le gain maximum pour distance connue.

Dans ce rapport, vous trouverez tout d'abord une présentation de notre solution. Plus précisément, nous présenterons un diagramme de classes ainsi qu’une brève description des implémentations de nos fonctions. Puis, nous discuterons des difficultés que nous avons rencontrées.

# Présentation de la solution

## Diagramme de classes

Afin de bien comprendre comment notre programme fonctionne, vous trouverez à la figure 1 le diagramme de classe de notre programme.

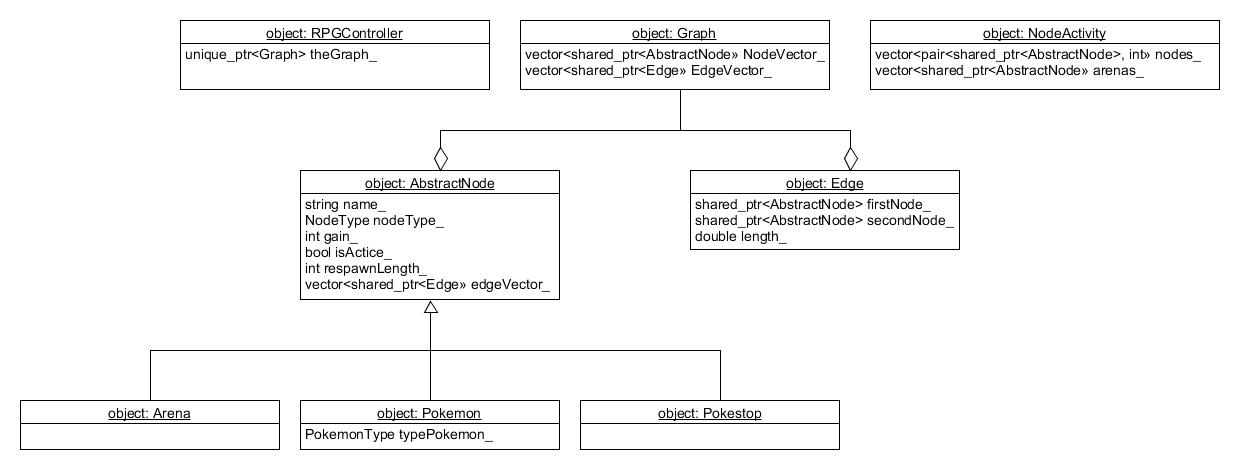


Figure 1 - Diagramme de classe

Nous avons décidé d’avoir une classe nommée *Graph* qui a un lien d’agrégation avec les classes *AbstractNode* et *Edge*. La classe *AbstractNode* est une classe abstraite qui sera dérivée en trois sous-classes ; Aréna, Pokémon et Pokéstop. De plus, nous avons la classe NodeActivity qui sert à s’occuper de rendre inactif les nœuds et de les remettre actifs une fois le nombre la distance à parcourir a été atteint. Finalement, nous avons la classe RPGController qui sert à appeler les fonctions de notre programme.

## Implémentation

### Fonction créerGraphe

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction lireGraphe

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction plusCourtChemin

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction plusGrandGain

Pour l’implémentation de cette classe, nous avions d’abord à nous questionner sur la façon où nous allions considérer qu’un nœud apporte un plus grand gain qu’un autre. Pour se faire, nous avons décidé que nous allions considérer les gains relatifs. Donc, toutes les fois que nous étions sur un nœud, nous regardons pour chacun de ses nœuds adjacents combien est le gain divisé par la distance à parcourir et nous prenons celui avec le plus grand nœud relatif comme prochain nœud. Nous répétons jusqu’à ce que nous avons atteint la distance demandée.

De plus, pour y parvenir, nous avons réalisé que nous allions fréquemment être à la recherche du nœud opposé au nœud courant. De ce fait, nous avons décidé de faire une méthode permettant d’obtenir le nœud opposé.

Finalement, lorsque nous visitons un nœud, il fallait le rendre inactif et puis les réactiver après un certain nombre de kilomètre marchés. Pour ce faire, nous avons implémenté une classe nommée *NodeActivity* qui contient un vecteur de pointeur vers les nœuds inactif ainsi que la distance restante à parcourir afin qu’il redevienne actif.

# Difficultés rencontrées

### Fonction créerGraphe

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction lireGraphe

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction plusCourtChemin

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

### Fonction plusGrandGain

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum

# Conclusion

iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum iorem ipsum