Travail Pratique 1

INF1008

Analyse et Conception d'algorithme

Présenté a

Adam Joly

Préparé par

Marius Couet [COUM77070000]

Nicolas Landry [LANN08069706]

Sviatoslav Besnard [BESS86080000]

Université du Québec à Trois-Rivières

Hiver 2020

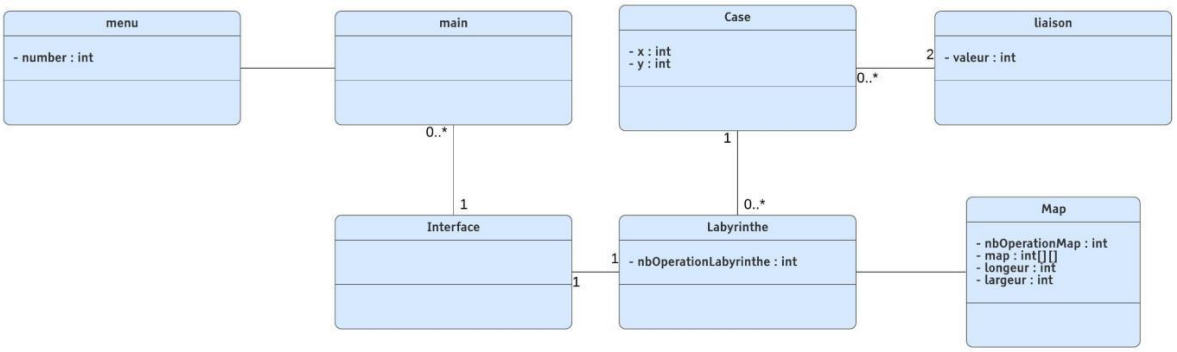
Table des matières

Application....2
Difficulté au niveau de l'algorithme....3
Difficulté au niveau de l'affichage....3
L'initialisation de la matrice d'adjacence :....4
La génération du labyrinthe et la lecture/l'affichage du labyrinthe :....4

Application

L'application a été code en Java.

Voici un diagramme de classe sans les interfaces (java), et simplifié.



Nous avons décidé de respecter le MVC, avec plusieurs classes dans chacun des packages

Dans le package controller (main dans notre application) :

- main
- menu

Dans le package model :

- Package data :
  - Map
- Package observer :
  - Observable (interface)
  - Observer (interface)
- Case
- Labyrinthe
- Liaison

Dans le package vue :

- Interface

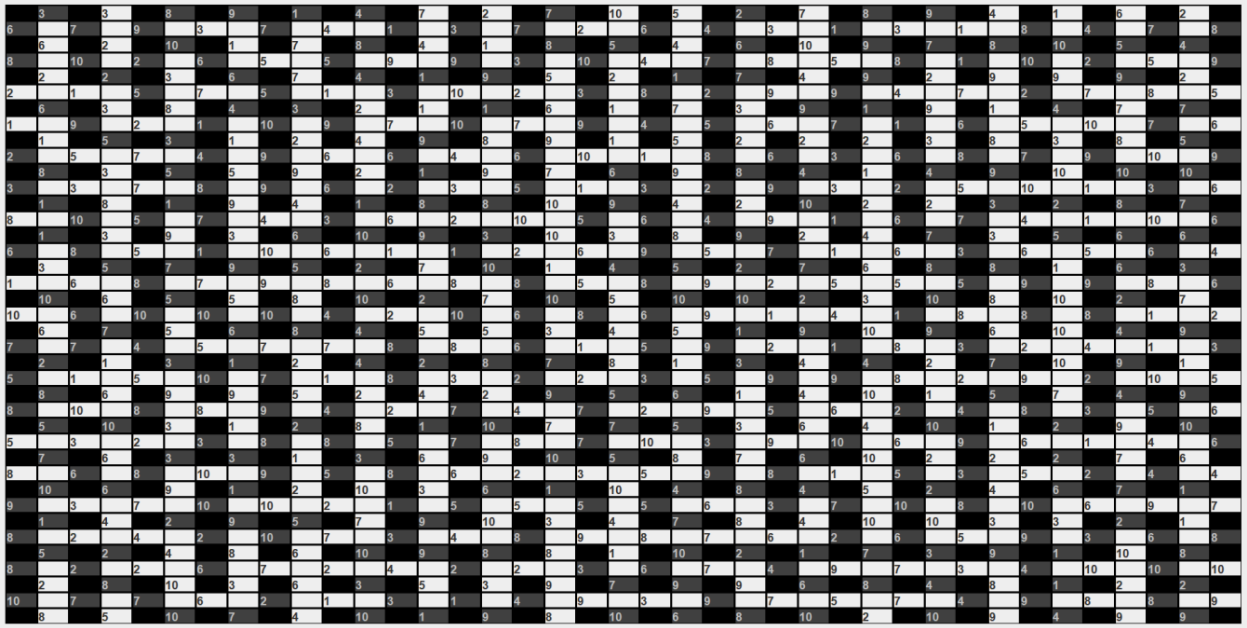
La classe main est la classe exécutée de base, elle utilise la classe menu pour génère l'interface.

L'interface génère le labyrinthe qui contient des cases qui elle-même ont des liaisons. Le Labyrinthe a une map.

Problème et difficulté rencontré

Difficulté au niveau de l'algorithme

Lors de la programmation de l'algorithme nous avons eu des difficultés avec les conditions ce qui nous donnais des chemins incohérents.



On peut voir que le chemin ne respecte pas un algorithme de Prim.

Difficulté au niveau de l'affichage

Malheureusement notre affichage n'est pas optimisé et donc il devient difficile voir impossible de voir les valeurs des liaisons lorsque le labyrinthe devient trop grand, et si le labyrinthe est encore plus grand il devient quasiment impossible de le généré avec l'affichage.

Opération

L'initialisation de la matrice d'adjacence :

```
public Map metDesValeursAleatoires(int min, int max){
    // x et y sont des coordonées en niveau du tableau map
    int x, y;
    for (x = 0; x < longueur*largeur; x++) {
        for (y = 0; y < 2; y++) {
            map[x][y] = (int) Math.floor((Math.random() * ((max - min) + 1)) + min);
            nbOperationMap += 1;
        }
    }
    return this;
}
```

La seule opération de boucle faites dans map (pour l'initialisation de la matrice d'adjacence) est celle-ci. Ce qui représente une complexité de $O(2n^2)$, soit $O(n^2)$.

La génération du labyrinthe et la lecture/l'affichage du labyrinthe :

```
for(int y=0; y<((largeur*2)-1); y++){
    for(int x=0; x<((longueur*2)-1); x++){
```

Cette boucle est la seul utilisé pour afficher le labyrinthe, on obtient donc par simplification une complexité de $O(n^2)$, car $O(4n^2)$ ce simplifie.

Prim

```
while (casesSolutions.size() < getMap().getLargeur() * getMap().getLongueur()) {
    casesSolutions.add(caseToAdd);

    // on ajoute les voisins de la case ajouté dans les liaisons possible
    for (Case.Direction direction : Case.Direction.values()) {
        // try si la case n'a pas de voisin dans la direction donnée
```

La génération du labyrinthe se fait avec ces boucles ci. La première représente une complexité de $O(n^2)$, et la seconde $O(4)$. Soit $O(n^2)$, car $O(4n^2)$ ce simplifie.

La complexité générale :

La complexité totale est une suite de complexité $O(n^2)$, donc l'algorithme a une complexité de $O(n^2)$.