Compte Rendu : Projet C++

4 ETI IMI

DUBOS Johanna – FENG WEI QIANG Jean pierre

Sommaire

I/ Introduction

L'objectif de ce projet est de développer une plateforme de démonstration de différents algorithmes d'exploration via le langage de programmation C++11 par l'intermédiaire de l'API Qt.

Ainsi nous allons partager ce compte rendu en quatre parties :

- La description de la fenêtre vue par le client
- La décomposition des différents fichiers qui le construisent
- Les algorithmes d'exploration mis en place
- Les problèmes rencontrés et améliorations possibles

II/ La description de la fenêtre vue par le client

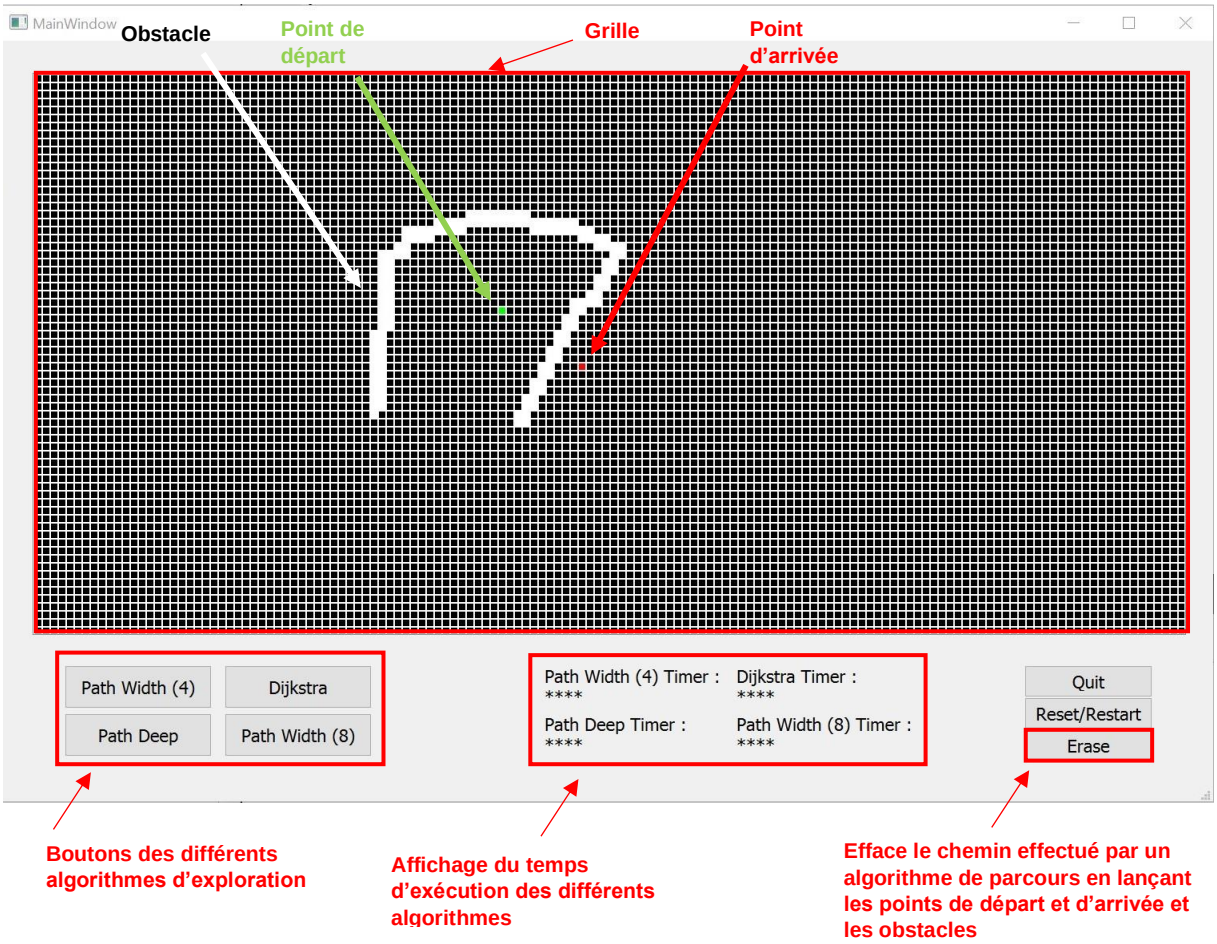

Figure 1 : Fenêtre de la plateforme

III/ Décomposition des différents fichiers

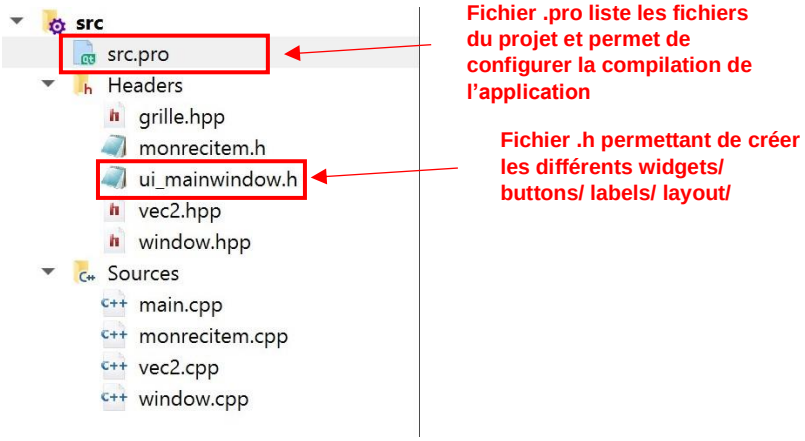

Figure 2 : Les différents fichiers

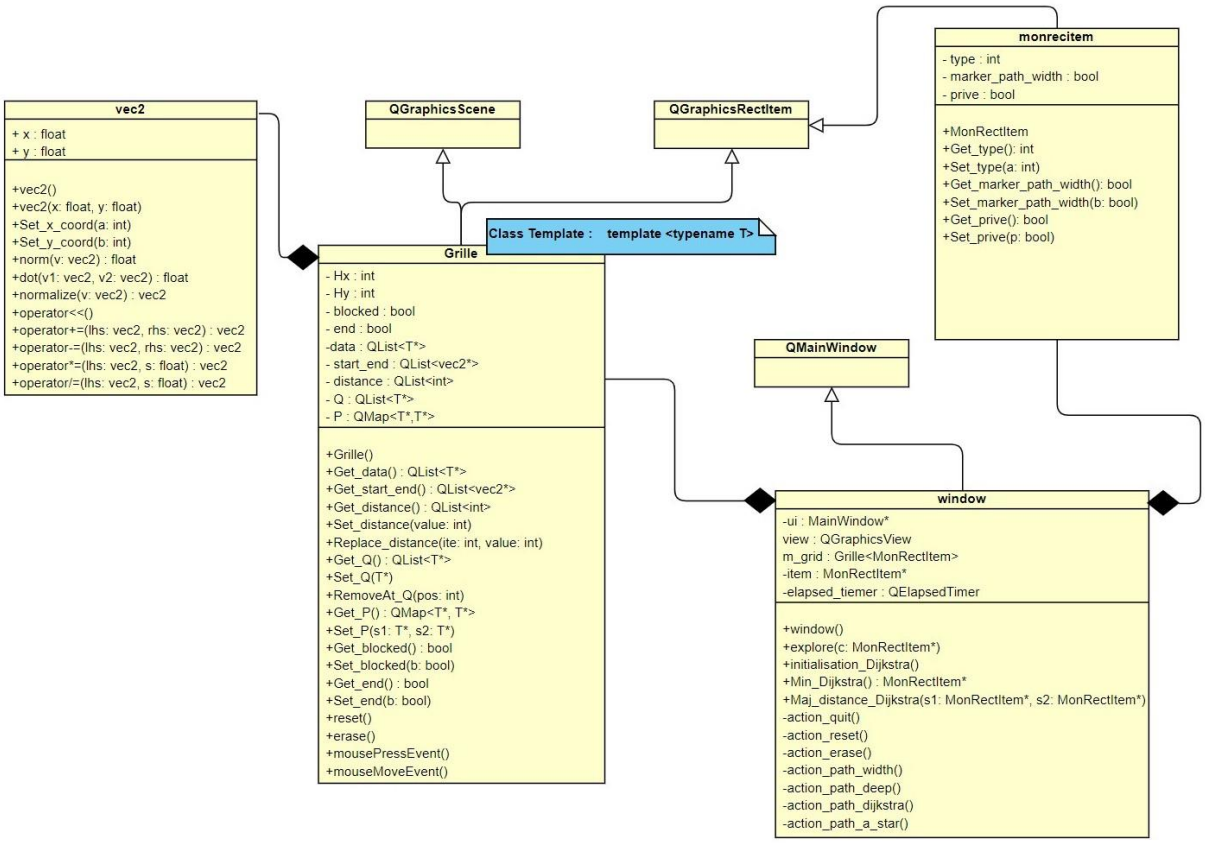

Figure 3 : Diagramme de classe

Grille :

Une classe template permettant de mettre en place la grille avec comme méthodes principales :

reset(), erase(), mosePressEvent(), mouseMoveEvent(), les getters et setters.

Monrecitem :

Une classe permettant de représenter les cases de la grille, notamment de savoir si elle est marquée et de quel type elle est (vide, point de départ, point d'arrivée, obstacle).

Vec2 :

Une classe permettant de représenter et manipuler des vecteurs à deux dimensions utilisés pour stocker les coordonnées des cases.

Window :

Une classe permettant de représenter la fenêtre d'affichage et notamment toutes les fonctions nécessaires pour l'action d'un bouton ainsi que les différentes fonctions permettant de coder les algorithmes d'exploration.

IV/ Algorithmes d'explorations

4.1. Parcours en largeur

Principe :

À partir de notre point de départ D, il faut tout d'abord lister les voisins de D pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une QList dans laquelle il prend notre point de départ et place en dernier ses voisins non encore explorés. Les cases déjà visitées sont marquées afin d'éviter qu'une même case ne soit explorée plusieurs fois.

Complexité :

La complexité en temps dans le pire cas est en θ(N) où N est le nombre de cases. En effet, chaque case est visitée au plus une seule fois.

4.2. Parcours en profondeur

Principe :

Pour chaque case, on marque la case actuelle, et on prend la première case voisine jusqu'à ce qu'une case n'ait plus de voisins (ou que tous ses voisins soient marqués ou qu'on rencontre un obstacle), et revient alors à la case père. L'exploration s'arrête quand on trouve un chemin conduisant à la case d'arrivée. On marque également les cases que l'on visite, de façon à ne pas les explorer à nouveau.

Complexité :

Même raisonnement que pour le parcours en largeur, le parcours en profondeur a une complexité dans le pire des cas de l'ordre θ(N) où N est le nombre de cases.

4.3. Dijkstra

Principe :

L'algorithme prend notre grille où chaque case est pondérée par un réel positif (nommé distance). Il s'agit de construire progressivement une Map dans laquelle sont classées les différentes cases par ordre croissant de leur distance minimale à la case de départ. La distance correspond à la somme des poids des arcs empruntés (ici on prendra un poids de 1 pour tous nos arcs).

Au départ, on considère que les distances de chaque sommet au sommet de départ sont infinies (ici nous avons mis une grande valeur : 2^{32}), sauf pour le sommet de départ pour lequel la distance est nulle. La Map de départ est l'ensemble vide.

Au cours de chaque itération, on choisit en dehors de la Map une case de distance minimale et on l'ajoute à la Map. Ensuite, on met à jour les distances des cases voisines de celles ajoutées.

On continue ainsi jusqu'à épuisement des sommets.

A la suite de cet algorithme, on obtient une Map contenant les différentes distances minimales avec leur prédécesseur. Ainsi avec cette Map, on remontera au chemin le plus court.

V/ Problèmes et améliorations possibles

Les différents bug et problèmes :

- Lors du dessin des obstacles, si la souris sort de la grille, l'interface s'arrête.
- Pour l'algorithme en profondeur, l'algorithme ne prend pas en compte les obstacles.
- Pour l'algorithme de Dijkstra, le temps d'exécution est très long (35s), et ne donne pas le chemin le plus court.

Améliorations possibles :

- Rajout d'une bar permettant de changer la dimension de la grille .
- Rajout d'une bar pour faire varier la vitesse de parcours .
- Affichage des cases une à une est non d'un seul coup.