Rapport du projet Starlight

Florian Knop (39310) - Gatien Bovyn (39189)

March 30, 2015

Table des matières

Introduction

Ce document vise à présenter le travail d'analyse et de programmation effectué lors de la réalisation du projet du laboratoire Langage C++ : Starlight.

Ce projet a été réalisé en binôme par Florian Knop, matricule 39310 groupe 2G13, et Gatien Bovyn, matricule 39189 groupe 2G11.

Le programme à concevoir consiste en une implémentation du modèle et d'une interface graphique du jeu baptisé Starlight, puzzle à 2 dimensions basé sur la lumière.

Sections

Présentation des différentes classes

L'implémentation du projet est divisée entre la partie modèle et la partie vue. Elle est également basée sur le design pattern Observeur / Observé comme demandé dans les consignes.

Modèle

Un squelette de l'application nous a été fourni par Monsieur Absil. Ce squelette contient les fichiers suivants :

'point.h, source.h, dest.h, nuke.h, wall.h, crystal.h, lens.h, mirror.h, ray.h, level.h'.

Line

Cette classe représente une droite, elle possède un point et un angle. Gràce à cela on peut trouver n'importe quel point de la droite gràce en ayant la distance entre le point d'origine et le point d'arrivée.

Dans les méthodes intersects(...) de la classe Line, un passage par pointeur de pointeur est fait car un pointeur est passé par valeur et lors de l'initialisation il ne pointera donc pas la même adresse mémoire que le pointeur d'origine. Si on passait donc par simple pointeur, notre pointeur copié aurait une bonne zone mémoire mais notre pointeur d'origine resterait à *nullptr*.

Ellipse

Cette classe représente une conique de forme elliptique. C'est à dire une ellipse ou un cercle.

La formule d'une ellipse est :

$$E \equiv ((x - x1)^2/a^2) + ((y - y1)^2/b^2) = 1$$

où x1 et y1 sont respectivement les coordonnées x et y du centre de l'ellipse. où a et b sont respectement les rayons de l'axe x et y.

Pour trouver une intersection entre une ellipse et une droite, il faut égaler deux variables identiques :

L'équation d'une droite non verticale est :

$$D \equiv y = ax + b$$

L'équation d'une droite verticale sera :

$$D \equiv x = k$$

où k est une valeur quelquonque sur l'axe des x.

On doit donc remplacer la variable x ou y de la droite dans l'équation de l'ellipse.

Le cas de la droite verticale :

Dans ce cas la, il n'y a pas de choix, il faut remplacer x dans l'équation de l'ellipse. Nous avons également décidé de refactoriser l'équation en prenant le PPCM (Plus petit commun multiple) de a^2 * b^2 que nous apellerons ici lcm pour Least Commun Multiple. Ce choix a été fait pour éviter les overflows lorsque de nombres trop grands sont mis au carré et multipliés. Bien que dans notre cas, on nous avons rarement des nombre pouvant obtenir un tel résultat.

$$E \equiv (lcmy * (k - x1)^2) + ((y - y1)^2 * lcmx) = lcm$$

où lcmy est le facteur par lequel il faut multiplier b^2 (rayon y au carré) pour obtenir lcm. où lcmx est le facteur par lequel il faut multiplier a^2 (raaon x au carré) pour obtenir lcm.

$$E \equiv (lcmy * (k - x1)^2 + (y^2 + y1^2 - 2 * y1 * y) * lcmx = lcm$$

$$E \equiv lcmy * (k - x1)^2 + lcmx * y^2 + lcmx * y1^2 - 2 * lcmx * y1 * y - lcm = 0$$

Avec ceci, il reste plus qu'à résoudre l'équation du second degré avec

$$rho = b^2 - 4 * a * c$$

où

$$a = lcmx$$

$$b = 2 * lcmx * y1 * y$$

$$c = (lcmy * (k - x1)^2) + (lcmx * y1^2) - lcm$$

On a donc trois possibilités :

$$rho < 0$$

: Pas d'intersections

$$rho = 0$$

: une seule intersection dont le y du point vaut : y = -b / (2 * a)

$$rho > 0$$

: deux intersections :

$$y1 = (-b + sqrt(rho))/(2 * a)$$

$$y2 = (-b - sqrt(rho))/(2 * a)$$

On a donc le(s) y du/des points d'intersections, et le x vaut k (de l'équation de départ).

Le cas de la droite non verticale :

C'est le même principe que le cas de la droite verticale sauf que pour trouver la deuxième variable finale il faudra remplacer la variable trouvée dans l'équation de la droite.

Vue

L'interface graphique a été réalisée en 'Qt' à la main.

Les classes composant la partie vue de l'application sont :

DestinationView

Description Classe modélisant la destination à atteindre par le rayon émis depuis la source pour gagner la partie.

MapView

MirrorView

NukeView

SourceView

WallView

Conclusion

Bibliographie

Annexes (facultatif)