Projet 3 : L'arbre des blocs (4 séances, à rendre)

1. Introduction

Le « monde des blocs » sert souvent de modèle simplifié pour l'analyse de problèmes d'intelligence artificielle. Ce qui nous intéresse ici en est une variante : des cubes de diverses dimensions sont empilés (certains sur d'autres) sur une table (une grosse caisse en fait) ; nous supposerons que ces cubes ont une longueur d'arête entière impaire et qu'un cube n'est jamais en « porte-à-faux », c'est à dire qu'il est totalement posé sur le cube (ou la table) dessous. Les coordonnées des sommets des cubes sont supposées « tomber juste » : ce sont des entiers, le centre de la table étant l'origine du repère.

On désire implémenter une structure de données permettant de simuler l'empilement des cubes. Compte tenu des contraintes, il n'est nécessaire de stocker pour chaque cube que la longueur d'une arête et les deux coordonnées dans le plan de son centre (comme si on regardait la scène du dessus). Mais pour matérialiser l'empilement, chaque cube est associé au cube qui le soutient ; vous devez donc élaborer une structure arborescente à partir des classes suivantes (code fourni sur Madoc) :

```
class point
{  public:
       int x,y;
       point(){x=0;y=0;}
       point(int x,int y)
         {this->x=x;this->y=y;}
};
```

```
class cube
{   private:  point milieu; int demiCote;
    public:
       cube(){};
       cube(point c,int cote){milieu=c;demiCote=cote/2;}
       const point centre() const {return milieu;}
       int cote() const {return 2*demiCote+1;}
};
```

```
class arbrecubes
{  private:
            struct _noeud { cube bloc; _noeud * fils, * frere; } ;
            _noeud * _racine;
   public:
       arbrecubes(std::string nomfic);

       const cube table() const {return _racine->bloc;}
       std::vector<cube> dessus(const cube & CC) const;
       const cube soutien(const cube & CC) const;

       void ajouter(const cube & CC) ;
       void supprimer(const cube & CC) ;

       const cube cubede(const point & M) const;
       int hauteur(const point & M) const;

};
```

2. Implémentation

Complétez le code fourni (sans ajouter d'attribut aux classes) compte tenu que :

Les cubes sont repérés par les coordonnées (x,y) du projeté sur le sol de leur centre et par la demi-longueur, entière, des arêtes (ainsi la longueur des arêtes est toujour impaire). La table est elle-même un cube (donc l'arbre n'est jamais vide).

Le père d'un nœud est le cube de dessous, le fils est l'un des cubes posés dessus (NULL s'il n'y en a pas) ; le pointeur frère permet de créer une liste chaînée des cubes posés sur le même père. Pour accélérer certaines méthodes, les fils sont triés par coordonnée x croissante (puis par y croissant en cas d'égalité) ; ajoutez la définition de l'opérateur de comparaison dans la classe point.

La méthode `soutien(cube CC)` fournit le cube qui supporte `CC` (son père dans l'arbre). La méthode `dessus(cube CC)` donne les cubes posés directement dessus `CC` (ses fils, mais pas ceux posés sur ses fils).

La méthode `ajouter(cube CC)` doit insérer le cube dans l'arbre de façon à respecter les contraintes : il sera intégralement posé sur son père et ses fils (son fils et les frères de son fils) seront posés sur lui ; vous supposerez que les cubes fournis permettent de respecter cette contrainte. La méthode `supprimer(cube CC)` doit enlever le cube de l'arbre de façon à conserver le respect des contraintes : ses fils se retrouveront sur son père.

La méthode `cubede(point M)` fournit le cube le plus haut dont le projeté (carré) sur le sol contient le point M ; c'est le cube qui serait mouillé par une goutte de pluie tombant aux coordonnées de M. La méthode `hauteur(point M)` fournit l'altitude du point d'impact d'une telle goutte (le sol est considéré à l'altitude `0`).

Le fichier texte fourni au constructeur `arbrecubes` doit présenter un cube sur chaque ligne : les deux coordonnées (x,y) du centre puis la dimension du côté (le reste de la ligne peut contenir des commentaires). La première ligne correspond à la table, les autres cubes sont donnés sans ordre particulier mais sont supposés respecter les contraintes du monde.

3. Rendu

Un document de travail doit être rédigé **au préalable et en parallèle**, précisant les ambiguïtés du sujet. Ce document présentera aussi les choix effectués (constantes, comportement des méthodes non précisé dans l'énoncé, …) et les limites d'utilisation. Il est aussi, si ce n'est plus important, que le code. Il devra comporter l'ordre de grandeur du coût de chaque méthode. L'implémentation proposée doit être discutée : en particulier quel est l'impact du tri des fils, comment pourrait-on modifier les classes pour accélérer le traitement ou diminuer l'empreinte mémoire ?

Le travail est à déposer sur MADOC au plus tard le 30 avril 2014 à 23h55 sous forme d'une archive zip contenant le rapport (pdf) et le code.

Le code doit être clair et concis, commenté, avec en particulier les pré- et post-conditions. Le rapport doit être bien écrit (français correct, attention à l'orthographe!) et agréable à lire, il doit comporter introduction et conclusion et des annexes pour les détails techniques (dessins, code, etc.)