Laboratorio 1

Estructura de datos

Juan Manuel López Vargas

Sebastián Almanza Galvis

David León Velásquez

Juan Sebastián Mendez

Fecha 31 de Julio 2024

1. Diseño:

Diseñe el sistema y el (los) TAD(s) solicitado(s). Utilice la plantilla de especificación de TADs vista en clase para el diseño. Recuerde que diseñar es un proceso previo a la implementación, por lo que no debería contener ninguna referencia a lenguajes de programación (es decir, si escribe encabezados o código fuente, el punto no será evaluado y tendrá una calificación de cero). Para simplicidad del diseño, no es necesario incluir los métodos obtener y fijar (get/set) del estado de cada TAD.

#TAD Punto

Estado:

- ' x ': R
- ' Y ': R
- ' Z ': R
- ' R ': R
- ' G ': R
- ' B ': R

Interfaz:

- `Punto (x, y, z, r, g, b) '
  - Post: Crea un punto con las coordenadas (x, y, z) y color (r, g, b)

- `ObtenerCoordenadas()`: (R, R, R)
  - Post: Retorna las coordenadas (x, y, z) del punto.

- `ObtenerColor()`: (R,G,B)
  - Pre:

    - $0 \leq r \leq 255$

    - $0 \leq g \leq 255$

- $0 \leq b \leq 255$

- Post: Retorna el color del punto en formato (r, g, b).

#TAD Nube de puntos

Estado:

- `nombre`: String
- `puntos`: List[Punto]

Interfaz:

- `NubeDePuntos(nombre)`
  - Post: Crea una nueva nube de puntos con el nombre `nombre`

- `AgregarPunto(punto: Punto)`
  - Post: Agrega un punto a la lista `puntos` de la nube

- `CalcularCentroide()`: Punto
  - Pre: `puntos` no vacío
  - Post: Retorna el centroide de la nube de puntos

- `ActualizarIndicadorDeVisualización(límitesEscena)`: String
  - Pre: `límitesEscena` es un objeto que define los límites de la escena.
  - Post: Retorna "completa", "parcial" o "nula" basado en la posición de los puntos respecto a los límites de la escena.

- `ObtenerNombre()`: String
  - Post: Retorna el nombre del objeto.

- `ObtenerPuntos()`: List[Punto]
  - Post: Retorna la lista de puntos en la nube.

#TAD Escenea

Estado:

- `límites`: (R, R, R, R, R, R)
- `objetos`: List[NubeDePuntos]

Interfaz:

- `Escena(límites)`
  - Post: Crea una escena con los límites `límites` en los tres ejes de coordenadas.

- `AgregarObjeto(objeto: NubeDePuntos)`
  - Post: Agrega un objeto (nube de puntos) a la lista `objetos`.

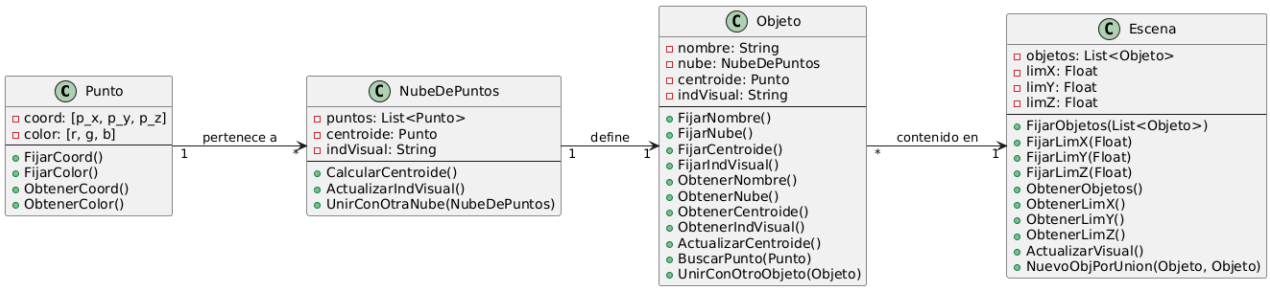
- `ActualizarIndicadorDeVisualización(objeto: NubeDePuntos)`
  - Post: Actualiza el indicador de visualización para el objeto dado en la escena.

- `CrearObjetoUnión(objeto1: NubeDePuntos, objeto2: NubeDePuntos)`: NubeDePuntos
  - Pre: `objetos` contiene `objeto1` y `objeto2`
  - Post: Crea y retorna un nuevo objeto resultante de la unión de `objeto1` y `objeto2`, calculando el centroide y asignando el indicador de visualización.

- `OrdenarObjetosPorCercanía()`
  - Post: Ordena los objetos en la escena por su cercanía al observador, de más lejano a más cercano.

- `DibujarEscena()`
  - Post: Dibuja la escena en pantalla respetando el orden de cercanía de los objetos.

1. Diagrama de relación:



2. Operación 1 – Actualizar indicador de visualización:

Prototipo de funciones y métodos:

```
#include <iostream>
#include <vector>

using namespace std;

class Punto {
public:
float coord [3]; // Coordenadas (x, y, z)
int color [3];
};

class NubeDePuntos{
public:
vector <Punto> puntos;
Punto centroide;
```

```
void CalculaCentroide();
string DeterminarIndicadorVisual (float x, float y, float z);
};

class Objeto{
public:
string nombre;
string inVisual;
NubeDePuntos nube;
Punto centroide;

void ActualizarIndicadoresVisual( float x, float y, float z);

};

#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Punto {
public:
   float coord[3]; // Coordenadas (x, y, z)
   int color[3];   // Color (r, g, b)

   float getX() const { return coord[0]; }
   float getY() const { return coord[1]; }
   float getZ() const { return coord[2]; }
};

class NubeDePuntos {
public:
   vector<Punto> puntos;
   Punto centroide;

   void CalculaCentroide() {
      // Implementación para calcular el centroide
```

```
    }

    string DeterminarIndicadorVisual(float minX, float maxX, float minY, float maxY,
float minZ, float maxZ) {
        bool completamenteDentro = true;
        bool parcialmenteDentro = false;

        for (const auto& p : puntos) {
            bool dentroDeLimites = (p.getX() >= minX && p.getX() <= maxX) &&
                                   (p.getY() >= minY && p.getY() <= maxY) &&
                                   (p.getZ() >= minZ && p.getZ() <= maxZ);

            if (!dentroDeLimites) {
                completamenteDentro = false;
            } else {
                parcialmenteDentro = true;
            }
        }

        if (completamenteDentro) {
            return "completa";
        } else if (parcialmenteDentro) {
            return "parcial";
        } else {
            return "nula";
        }
    }
};

class Objeto {
public:
    string nombre;
    string inVisual;
    NubeDePuntos nube;
    Punto centroide;

    void ActualizarIndicadoresVisual(float minX, float maxX, float minY, float maxY,
float minZ, float maxZ) {
        inVisual = nube.DeterminarIndicadorVisual(minX, maxX, minY, maxY, minZ,
maxZ);
```

```
    }
};

int main() {
    // Ejemplo de uso
    Objeto obj;
    obj.nube.puntos.push_back({{1.0, 2.0, 3.0}, {255, 0, 0}});
    obj.nube.puntos.push_back({{4.0, 5.0, 6.0}, {0, 255, 0}});

    // Definir los límites de la escena
    float minX = 0.0, maxX = 5.0, minY = 0.0, maxY = 5.0, minZ = 0.0, maxZ = 5.0;

    obj.ActualizarIndicadoresVisual(minX, maxX, minY, maxY, minZ, maxZ);

    cout << "Indicador Visual: " << obj.inVisual << endl;

    return 0;
}
```

3. Operación 2 - Unión de dos objetivos:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

class Punto {
public:
    float coord[3]; // Coordenadas (x, y, z)
    int color[3];   // Color (r, g, b)

    float getX() const { return coord[0]; }
    float getY() const { return coord[1]; }
    float getZ() const { return coord[2]; }
};

class NubeDePuntos {
```

```
public:
   vector<Punto> puntos;
   Punto centroide;

   void CalculaCentroide() {
      float sumaX = 0.0, sumaY = 0.0, sumaZ = 0.0;
      int numPuntos = puntos.size();
      for (const auto& p : puntos) {
         sumaX += p.getX();
         sumaY += p.getY();
         sumaZ += p.getZ();
      }
      centroide.coord[0] = sumaX / numPuntos;
      centroide.coord[1] = sumaY / numPuntos;
      centroide.coord[2] = sumaZ / numPuntos;
   }

   string DeterminarIndicadorVisual(float minX, float maxX, float minY, float maxY,
float minZ, float maxZ) {
      bool completamenteDentro = true;
      bool parcialmenteDentro = false;

      for (const auto& p : puntos) {
         bool dentroDeLimites = (p.getX() >= minX && p.getX() <= maxX) &&
                                (p.getY() >= minY && p.getY() <= maxY) &&
                                (p.getZ() >= minZ && p.getZ() <= maxZ);

         if (!dentroDeLimites) {
            completamenteDentro = false;
         } else {
            parcialmenteDentro = true;
         }
      }

      if (completamenteDentro) {
         return "completa";
      } else if (parcialmenteDentro) {
         return "parcial";
      } else {
         return "nula";
```

```
        }
    }
};

class Objeto {
public:
    string nombre;
    string inVisual;
    NubeDePuntos nube;
    Punto centroide;

    void ActualizarIndicadoresVisual(float minX, float maxX, float minY, float maxY,
float minZ, float maxZ) {
        inVisual = nube.DeterminarIndicadorVisual(minX, maxX, minY, maxY, minZ,
maxZ);
    }
};

Objeto UnirObjetos(const Objeto& obj1, const Objeto& obj2, float minX, float maxX,
float minY, float maxY, float minZ, float maxZ) {
    Objeto nuevoObjeto;
    nuevoObjeto.nombre = obj1.nombre + "_" + obj2.nombre;

    // Unir las nubes de puntos
    for (const auto& p : obj1.nube.puntos) {
        nuevoObjeto.nube.puntos.push_back(p);
    }

    for (const auto& p : obj2.nube.puntos) {
        auto it = find_if(nuevoObjeto.nube.puntos.begin(), nuevoObjeto.nube.puntos.end(),
[&p](const Punto& punto) {
            return (punto.getX() == p.getX() && punto.getY() == p.getY() && punto.getZ()
== p.getZ());
        });
        if (it == nuevoObjeto.nube.puntos.end()) {
            nuevoObjeto.nube.puntos.push_back(p);
        }
    }

    // Calcular el centroide del nuevo objeto
```

```
    nuevoObjeto.nube.CalculaCentroide();
    nuevoObjeto.centroide = nuevoObjeto.nube.centroide;

    // Actualizar el indicador visual del nuevo objeto
    nuevoObjeto.ActualizarIndicadoresVisual(minX, maxX, minY, maxY, minZ, maxZ);

    return nuevoObjeto;
}

int main() {
    // Ejemplo de uso
    Objeto obj1;
    obj1.nombre = "Objeto1";
    obj1.nube.puntos.push_back({{1.0, 2.0, 3.0}, {255, 0, 0}});
    obj1.nube.puntos.push_back({{4.0, 5.0, 6.0}, {0, 255, 0}});

    Objeto obj2;
    obj2.nombre = "Objeto2";
    obj2.nube.puntos.push_back({{4.0, 5.0, 6.0}, {0, 255, 0}});
    obj2.nube.puntos.push_back({{7.0, 8.0, 9.0}, {0, 0, 255}});

    // Definir los límites de la escena
    float minX = 0.0, maxX = 10.0, minY = 0.0, maxY = 10.0, minZ = 0.0, maxZ = 10.0;

    Objeto nuevoObjeto = UnirObjetos(obj1, obj2, minX, maxX, minY, maxY, minZ,
maxZ);

    cout << "Nombre del nuevo objeto: " << nuevoObjeto.nombre << endl;
    cout << "Indicador Visual del nuevo objeto: " << nuevoObjeto.inVisual << endl;

    return 0;
}
```