En esta ocasión se completo el codigo del R tree que no s dio la miss, solo se realizaron cambios de nombres de unas partes para que cuadren con el paper, pero nada fuera de normal,

los cambios fueron

correción en esta frma de función por que generaba errores

bool Overlap(Rect* a_rectA, Rect* a_rectB) const;
->
bool Overlap(const Rect* a_rectA, const Rect* a_rectB) const;

Funciones Completadas y Explicación

listan las funciones clave que fueron completadas

1. FUNCION: bool RTree::Overlap(const Rect* a_rectA, const Rect* a_rectB) const

nos dice si los dos Rectángulos Delimitadores Mínimos proporcionados tienen alguna área en común. Esta es la base para la búsqueda y la inserción.

La función verifica para cada dimensión (X y Y) si hay una separación total. Si el límite máximo de un rectángulo es menor que el límite mínimo del otro en cualquier dimensión, se confirma que no hay superposición

2. FUNCION: void RTree::Search(const Rect& a_rect, vector<vector<pair<int, int>>>& a_results)

Es la función que busca .

Simplemente actúa como un wrapper o envoltorio, llamando a la función recursiva serchrec

3. FUNCION: void RTree::SearchRec(Node* a_node, const Rect& a_rect, vector<vector<pair<int, int>>>& a_results)

Implementa el Algoritmo serch de guttman de forma recursiva, recorriendo el árbol para encontrar los objetos que se superponen con la región de consulta.

Funcionamiento:

- **En Nodos Internos (No Hoja):** Itera sobre cada rama que apunta a un nodo hijo. Utiliza la función overlap para verificar si el MBR de la rama se superpone con la consulta. Si hay superposición, la función se llama a sí misma de forma recursiva en

el nodo hijo correspondiente. Esto permite podar grandes secciones del árbol que no son relevantes.

- **En Nodos Hoja:** Itera sobre cada objeto de datos. Utiliza overlap para verificar si el MBR del objeto individual se superpone con la consulta. Si se superpone, el objeto de datos original (m_data, que es el polígono) se añade al vector de resultados.

```
bool RTree::Search(const Rect& a_rect, vector<vector<pair<int, int>>>& a_results)
{
    a_results.clear();
    SearchRec(m_root, a_rect, a_results);
    return !a_results.empty();
}


void RTree::SearchRec(Node* a_node, const Rect& a_rect, vector<vector<pair<int, int>>>& a_results)
{

    if (a_node->IsInternalNode())
    {
        for (int index = 0; index < a_node->m_count; ++index)
        {
            if (Overlap(&a_node->m_branch[index].m_rect, &a_rect))
            {
                SearchRec(a_node->m_branch[index].m_child, a_rect, a_results);
            }
        }
    }
    else
    {
        for (int index = 0; index < a_node->m_count; ++index)
        {
            if (Overlap(&a_node->m_branch[index].m_rect, &a_rect))
            {
                a_results.push_back(a_node->m_branch[index].m_data);
            }
        }
    }
}
```

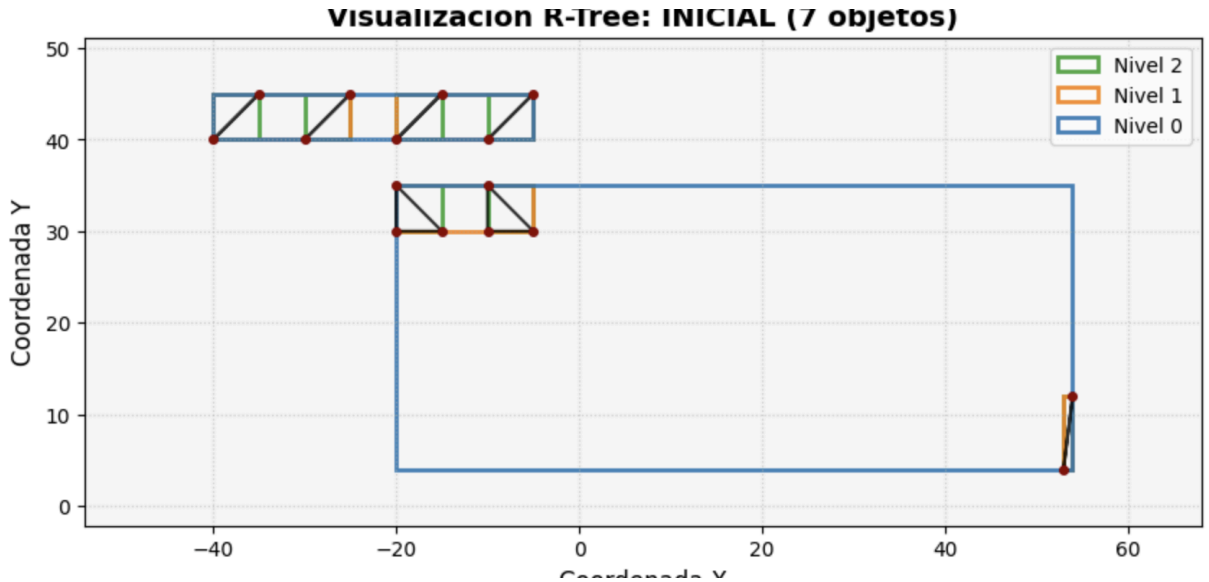
aqui se ve loq ue digo, recuersivamente y ademas ahi vemos si existe este solapamiento que estamos buscando

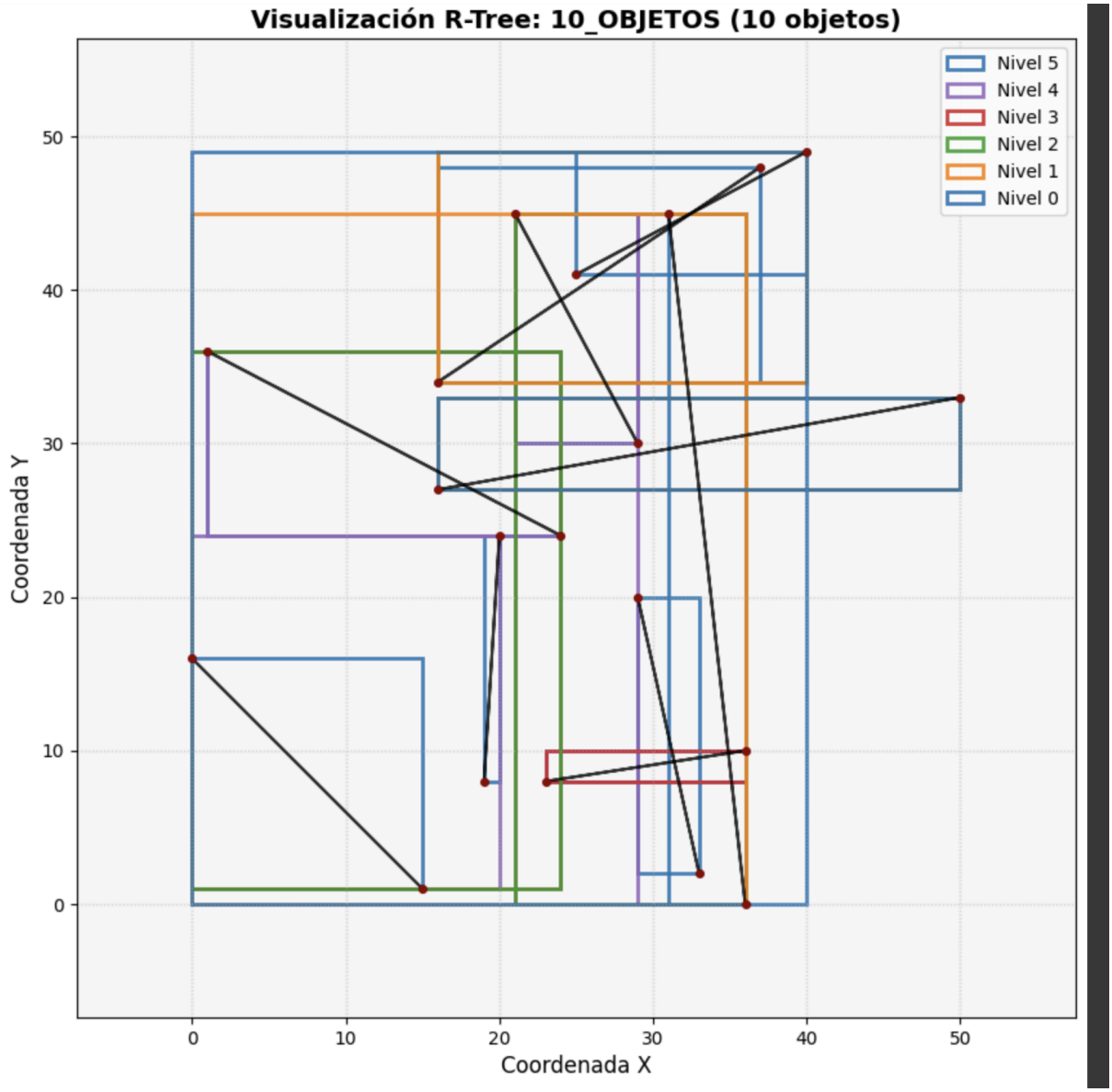
4. FUNCION: bool RTree::getMBRs(vector<vector<vector<pair<int, int>>>>& mbrs_n)

Esencial para la visualización. Recorre el árbol y extrae las coordenadas de todos los MBRs, organizándolos por nivel.

Realiza una recorrida por niveles, comenzando desde la raíz. Recolecta el par de puntos (coordenadas mínima y máxima) que define cada MBR en ese nivel. Los resultados se organizan en un vector tridimensional donde el primer índice representa el nivel del árbol, facilitando la graficación en Python.

ademas entre los cambios, habia una función que s ellamba PickBranch y la cambie para que se llame como la del paper ChooseLeaf, tambien ChoosePartition a QuadraticSplit y RemoveRectRec a DeleteRec, luego si mal no me acuero lo otro si lo deje




Visualización R-Tree: 10_OBJETOS (10 objetos)
Nivel 5
Nivel 4
Nivel 3
Nivel 2
Nivel 1
Nivel 0
Coordenada Y
Coordenada X
0
10
20
30
40
50