Complejidad Temporal, Estructuras de datos y Algoritmos

Trabajo Final

“Conquista planetaria”

Segundo Cuatrimestre 2023

Profesor: Leonardo Amet

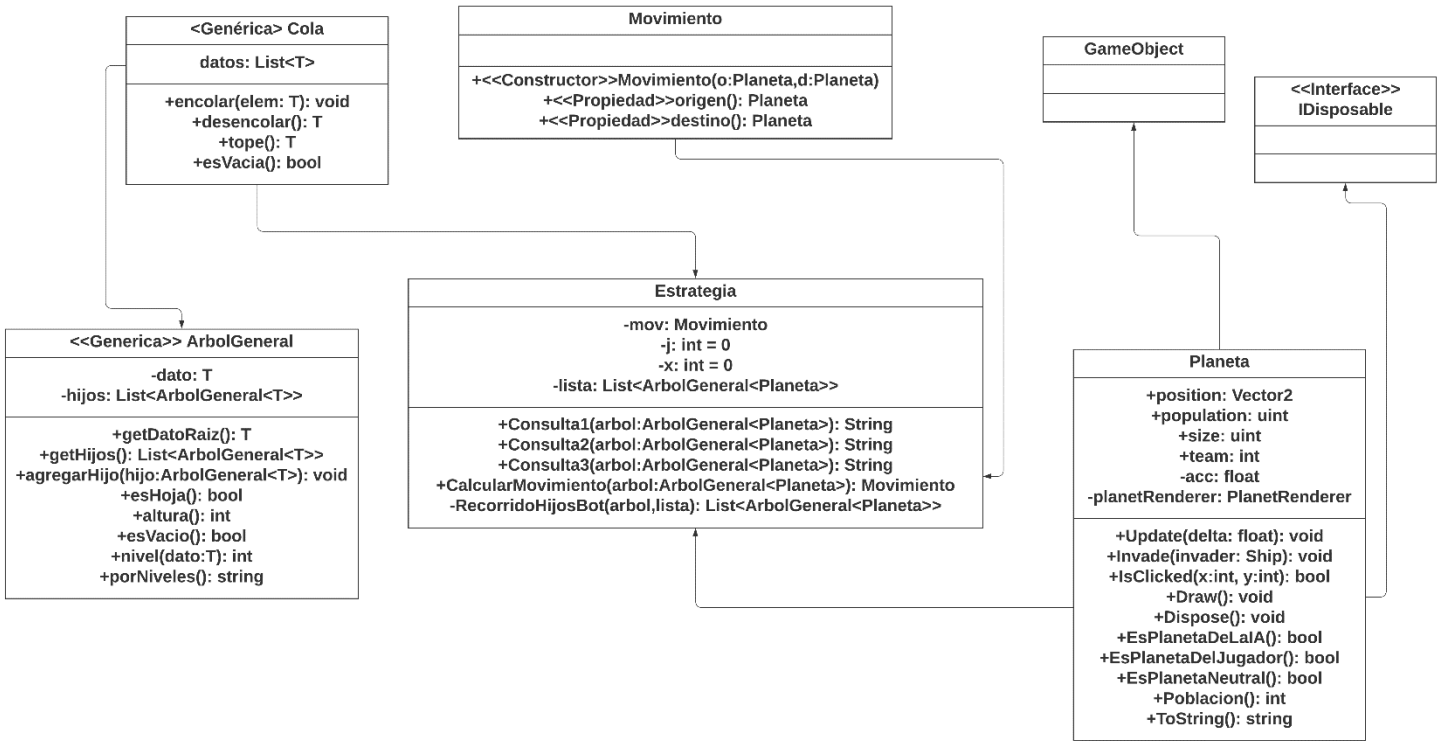
Comisión: 5

Estudiante: Mendes Ana

Introducción

El informe tiene como propósito principal exponer el progreso del trabajo práctico final de la asignatura Complejidad temporal, estructura de datos y algoritmos. En el marco de este proyecto, se procedió a la codificación de un juego centrado en la “conquista planetaria”, caracterizado por un enfrentamiento exclusivo entre dos jugadores.

Diagrama de clases UML



Detalles de la codificación

ESTRATEGIA DEL JUEGO

La **estrategia del juego** funciona detectando el planeta del bot, conquistando todos los planetas descendientes y reagrupando en el bot las naves nuevas en los planetas conquistados.

Tiene dos métodos:

- **public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol):**
  Este método calcula y retorna el movimiento apropiado según el estado del juego.
  El estado del juego es recibido en el parámetro árbol de tipo ArbolGeneral<Planeta>.
  Cada nodo del árbol contiene como dato un planeta del juego.
  Se inicia creando una cola, en la cual se encola el árbol original. Inicio un bucle y mientras la cola no esté vacía, se desencola un árbol de la cola, se verifica si la raíz del árbol es un planeta perteneciente a la IA. Si la raíz es planeta de la IA, se recorre de los hijos de ese árbol para construir una lista de árboles que representan los posibles movimientos del bot. Se examina la lista para determinar el próximo movimiento del bot, si se encuentra un movimiento válido, se crea un movimiento y se retorna. Si no se encuentra movimiento en el recorrido de los hijos se rompe el bucle. Si el nodo actual

no es planeta de la IA, se encola todos los hijos del nodo actual. Se ejecuta bucle secundario para intentar encontrar movimiento, si no se encuentra se retorna null.

```
private static Movimiento mov;
private static int j = 0, x = 0;
private static List<ArbolGeneral<Planeta>> lista = new List<ArbolGeneral<Planeta>>();
1 referencia
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> arbolAux;
    c.encolar(arbol);

    while (!c.esVacia())
    {
        arbolAux = c.desencolar();
        if (arbolAux.getDatoRaiz().EsPlanetaDeLaIA())
        {

            lista = RecorridoHijosBot(arbolAux, new List<ArbolGeneral<Planeta>>());
            while (j < lista.Count - 1)
            {
                if (!lista[j + 1].getDatoRaiz().EsPlanetaDeLaIA())
                {
                    if (!lista[j + 1].getDatoRaiz().EsPlanetaDeLaIA() && lista[j + 1].esHoja())
                    {

                        mov = new Movimiento(lista[j].getDatoRaiz(), lista[j + 1].getDatoRaiz());
                        j += 1;
                        return mov;
                    }
                    mov = new Movimiento(lista[0].getDatoRaiz(), lista[j + 1].getDatoRaiz());
                    j += 1;
                    return mov;
                }
            }
            break;
        }
        else
        {
            if (arbolAux.getHijos().Count != 0)
                foreach (var hijo in arbolAux.getHijos())
                    c.encolar(hijo);
        }
    }
```

```
    //Se ejecuta bucle secundario para intentar encontrar MOVIMIENTO, si no se encuentra se retorna NULL
    int t = (lista.Count - 1) - x;
    while(t > 0)
    {
        if(lista[t].esHoja())
        {
            mov = new Movimiento(lista[t].getDatoRaiz(), lista[t - 1].getDatoRaiz());
            x += 1;
            return mov;
        }
        mov = new Movimiento(lista[t].getDatoRaiz(), lista[0].getDatoRaiz());
        x += 1;
        return mov;
    }
    return null;
}
```

- **private List<ArbolGeneral<Planeta>> RecorridoHijosBot(ArbolGeneral<Planeta> arbol, List<ArbolGeneral<Planeta>> lista):**
  Este método recursivo recibe el árbol como parámetro y una lista donde se guardan los hijos, se utiliza una lista auxiliar que servirá de ayuda. Se agrega el árbol recibido a la lista del parámetro, si el árbol es una hoja se retorna la lista. SI el árbol no es una hoja, se recorre la lista de hijos del árbol, utilizando una lista auxiliar para realizar recursivamente al método, pasando al árbol como elemento actual de la iteración. Se verifica si la lista auxiliar es distinta de null y se retorna la lista auxiliar.

```
//Recorrido en profundidad de los hijos de un árbol de planetas y construye una lista de árboles
2 referencias
private List<ArbolGeneral<Planeta>> RecorridoHijosBot(ArbolGeneral<Planeta> arbol, List<ArbolGeneral<Planeta>> lista)
{
    List<ArbolGeneral<Planeta>> listaAux = null;
    lista.Add(arbol);

    if (arbol.esHoja())
        return lista;
    else
    {
        foreach (var item in arbol.getHijos())
            listaAux = RecorridoHijosBot(item, lista);

        if (listaAux != null)
            return listaAux;
    }

    return null;
}
```

CONSULTAS

Mediante un recorrido por niveles se implementan **las consultas 1, 2 y 3**, utilizando una Cola y un separador por nivel (null).

- **Consulta 1 (ArbolGeneral<Planeta> árbol)**: Calcula y retorna un texto con la distancia del camino que existe entre el planeta del Bot y la raíz.
  Suponiendo que el árbol pasado como parámetro corresponde al nodo raíz, se encola el árbol raíz y se encola un marcador de nivel null.
  Si el arbolAux es nulo, se incrementa el contador de nivel y se encola otro marcador null.
  Si arbolAux no es nulo y su raíz es un planeta de la IA, se actualiza el nivel. Si arbolAux no es nulo y tiene hijos, se encolan los hijos. Se retorna por último un mensaje indicando la distancia entre el planeta del bot y la raíz.

- **Consulta 2 (ArbolGeneral<Planeta> árbol)**: Retorna un texto con el listado de los planetas ubicados en todos los descendientes del nodo que contiene al planeta del Bot.
  Se encola el árbol raíz, se encola un marcador de nivel null. si el arbolAux es nulo se encola otro marcador null si la cola no está vacía.
  Si el arbolAux no es nulo y su raíz es un planeta de la IA, se obtiene la cadena de descendientes y se rompe el bucle.
  Si el arbolAux no es nulo y tiene hijos, se encolan los hijos. Se retorna por último un mensaje que incluye el planeta del bot y la lista de descendientes.

- **Consulta 3 (ArbolGeneral<Planeta> árbol)**: Calcula y retorna en un texto la población total y promedio por cada nivel del árbol.
  Se encola el árbol raíz, se encola un marcador de nivel nulo.
  Si el arbolAux es nulo, se incrementa el contador de nivel, se actualiza la población por nivel y se agrega a la cadena. Se encola otro marcador nulo si la cola no está vacía, se reinicia el contador de nodos para el próximo nivel.
  Si arbolAux no es nulo y no tiene hijos, se incrementa la población el nivel actual.
  Si arbolAux no es nulo y tiene hijos, se incrementa la población y se encolan los hijos.
  Se calcula el promedio de población por nivel y se retorna la cadena que contiene la información de población por nivel, población total y promedio.

```
//CONSULTA1: CALCULAR Y RETORNAR UN TEXTO CON DISTANCIA DEL CAMINO ENTRE PLANETAD EL BOT Y RAIZ
1 referencia
public String Consulta1(ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> arbolAux;
    int contNiv = 0, nivel = 0;
    c.encolar(arbol);
    c.encolar(null);
    while (!c.esVacia())
    {
        arbolAux = c.desencolar();
        if (arbolAux == null)
        {
            contNiv++;
            if (!c.esVacia())
                c.encolar(null);
        }
        if (arbolAux != null && arbolAux.getDatoRaiz().EsPlanetaDeLaIA())
            nivel = contNiv;
        else
        {
            if (arbolAux != null && arbolAux.getHijos().Count != 0)
            {
                foreach (var hijo in arbolAux.getHijos())
                    c.encolar(hijo);
            }
        }
    }
    // Se retorna un mensaje indicando la distancia entre el planeta del Bot y la raíz
    return $"* EXISTE ENTRE PLANETA DEL BOT Y RAÍZ UNA DISTANCIA DE: {nivel.ToString()}";
}
```

```
//CONSULTA2: RETORNA TEXTO CON LISTADO DE PLANETAS EN TODOS LOS DESCENDIENTES DEL NODO QUE CONTIENE
// AL PLANETA DEL BOT
1 referencia
public String Consulta2(ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> arbolAux = null;
    string cadena = " ";
    c.encolar(arbol);
    c.encolar(null);
    while (!c.esVacia())
    {
        arbolAux = c.desencolar();
        if (arbolAux == null)
        {
            if (!c.esVacia())
                c.encolar(null);
        }
```

```
            if (arbolAux != null && arbolAux.getDatoRaiz().EsPlanetaDeLaIA())
            {
                cadena = arbolAux.porNiveles();
                break;
            }
            else
            {

                if (arbolAux != null && arbolAux.getHijos().Count != 0)
                {
                    foreach (var hijo in arbolAux.getHijos())
                        c.encolar(hijo);
                }
            }
        }
        // Se retorna un mensaje que incluye el planeta del Bot y la lista de descendientes
        return "\n* PLANETA DEL BOT: " + "[" + arbolAux.getDatoRaiz() + "]" + "\n" +
                " DESCENDIENTES DEL BOT: " + cadena;
    }
```

```
//CONSULTA3: CALCULA Y RETORNA LA POBLACIÓN TOTAL Y PROMEDIO POR CADA NIVEL DEL ÁRBOL
1 referencia
public String Consulta3(ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> arbolAux;
    int contNiv = 0, cantNod = 0, contNodosNivel = 0;
    string cadena = "\n\n\n* POBLACIÓN POR NIVELES: \n";
    c.encolar(arbol);
    c.encolar(null);
    while (!c.esVacia())
    {
        arbolAux = c.desencolar();
        if (arbolAux == null)
        {
            contNiv++;
            contNodosNivel += cantNod;
            cadena += " NIVEL: " + contNiv + " POBLACIÓN: " + cantNod + "\n";
            if (!c.esVacia())
                c.encolar(null);
            cantNod = 0;
        }
```

```
        }
        if (arbolAux != null && arbolAux.getHijos().Count == 0)
        {
            cantNod++;
        }
        if (arbolAux != null && arbolAux.getHijos().Count != 0)
        {
            cantNod++;
            foreach (var hijo in arbolAux.getHijos())
                c.encolar(hijo);
        }
    }
    // Se calcula el promedio de población por nivel
    double prom = contNodosNivel / contNiv;
    // Se retorna la cadena que contiene la información de población por nivel, población total y promedio
    return cadena += "\n POBLACIÓN TOTAL DEL ÁRBOL: " + contNodosNivel + "\n"
                  + " PROMEDIO DE POBLACIÓN POR NIVEL: " + prom;
}
```

Posibles problemas

- Tardanza en el cálculo del movimiento.
- Una vez que se inicia el juego, no se puede volver al menú principal.
- No nos indica cual es nuestro color y el del enemigo al comenzar el juego.
- El bot a veces no comienza a distribuir sus naves fabricadas, solo las junta en un nodo.
- Nos deja tener en todos los nodos alrededor del oponente naves fabricándose.
- Es muy sencillo el juego, se podría complejizar.

Conclusión

A partir de la realización del trabajo final, destaco haber utilizado conceptos de la materia:

- Implementación de árboles generales
- Recorrido por niveles
- Cola

La cátedra nos proporcionó recursos que nos simplificaron mucho el trabajo. No obstante, fortalecí mi comprensión de los conceptos anteriormente punteados.