Trabajo Final de Complejidad Temporal, Estructura de Datos y Algoritmos

Alumno: Taiel Ojeda Studer

Comisión 5

Legajo: 67968

Profesor: Ing. Leonardo Javier Amet

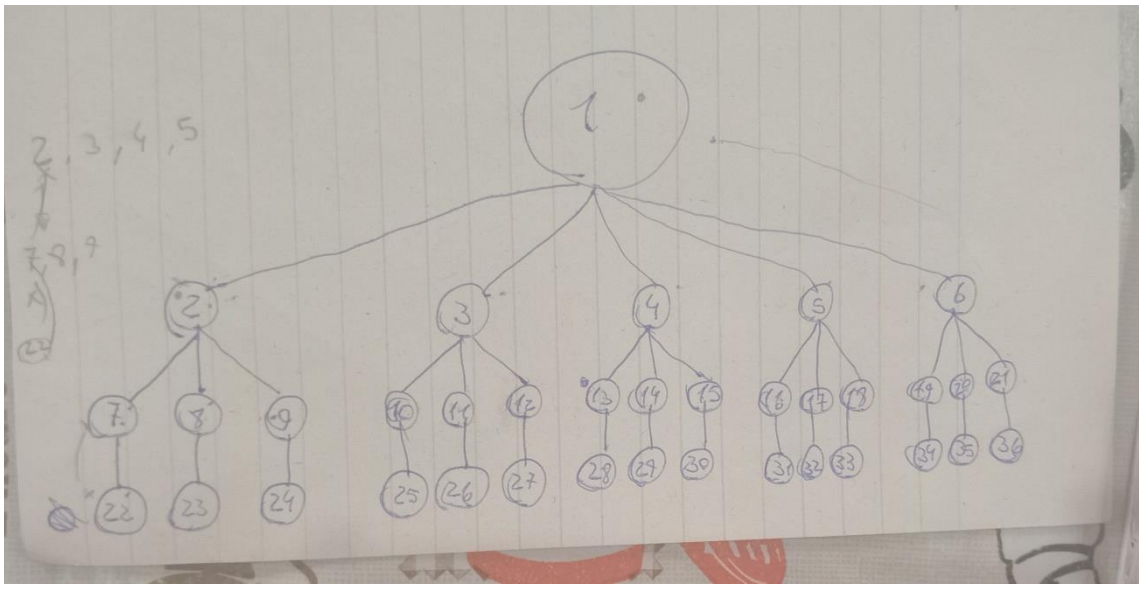
INDICE

Detalles de implementación

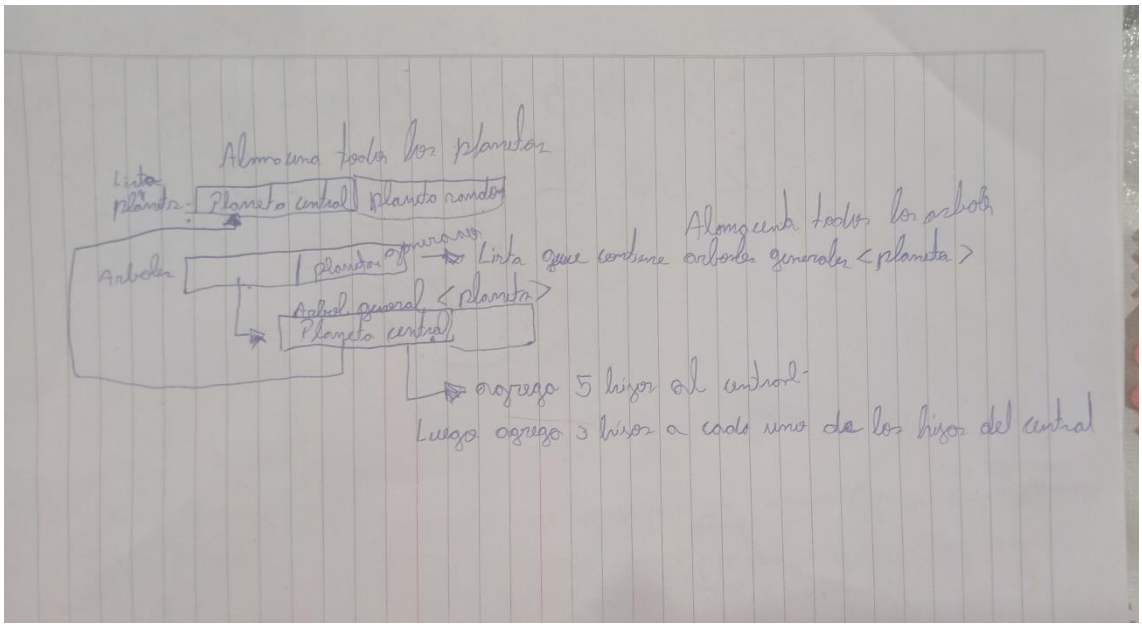
El trabajo presentó cierto tipo de complejidad que no había pasado en experiencias pasadas programando.
Lo novedoso del trabajo fue tener que programar sobre un proyecto dado por la cátedra la cual yo desconocía su funcionamiento. Lo primero que hice fue hacer un análisis superficial sobre el funcionamiento de la creación de niveles y árboles de Planetas. Para ello me valí de ayudas visuales.

	arboles	posicion	hijos de 1	hijos de 2	hijos de 3	hijos de 4	hijos de 5	hijos de 6
raiz	1	0						
1er nivel	2	1						
	3	2						
	4	3						
	5	4						
	6	5						
2do nivel	7	6						
	8	7						
	9	8						
	10	9						
	11	10						
	12	11						
	13	12						
	14	13						
	15	14						
	16	15						
	17	16						
	18	17						
	19	18						
	20	19						
	21	20						
3er nivel	22	21						
	{	22						
	24	23						
	25	24						
	26	25						
	27	26						
	28	27						
	29	28						
	30	29						
	31	30						
	32	31						
	33	32						
	34	33						
	35	34						
	36	35						

Este cuadro representa de manera superficial cómo se acomodan los nodos en una lista de árboles generales que contiene todos los árboles del sistema. A su vez también hay una lista que contiene todos los planetas del sistema, no les he encontrado alguna utilidad, por lo que descubrir esto no fue de ayuda para la solución de la consigna, más sí para entender sobre qué estoy trabajando.



También se hizo un boceto en papel de cómo se vería el árbol general del juego si estuviera posicionado de la forma más “común” para facilitar su análisis.



Por último, también se hizo un boceto a mano de cómo funciona superficialmente el algoritmo que da origen a los nodos Planetas.

Implementación de métodos

El primer método que se implementó fue el de Consulta 1. Este pudo lograrse a partir de diseñar también un método auxiliar que ayudara a calcular la profundidad del árbol general de forma recursiva.

```
public int Profundidad(ArbolGeneral<Planeta> arbol)
{
    if (arbol.getDatoRaiz().team == 2)
    {
        return 0;
    }
    foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
    {
        int profundidadEnHijo = Profundidad(hijo);
        if(profundidadEnHijo>=0)
        {
            return profundidadEnHijo+1;
        }
    }
    return -1;
}
```

La implementación se ve en el método Consulta 1:

```
public String Consulta1( ArbolGeneral<Planeta> arbol)
{
    int camino = Profundidad(arbol);

    return "El camino es de longitud"+camino;
}
```

Luego se implementó el método de calcular movimiento. Para ello fue necesario dos métodos auxiliares.

```
public int ComprobarHijos(ArbolGeneral<Planeta> arbol)
{

    int posicion = -1;
    for(int i = 0; i<=arbol.getHijos().Count-1;i++)
    {
        if (arbol.getHijos()[i].getDatoRaiz().team==2)
        {
            posicion = i;
            return posicion;
        }
    }
    return posicion;
}
```

Por un lado este método existe sólo para no complejizar la implementación de CalcularMovimiento ya que obligaría a anidar más ciclos for de lo deseado. Devuelve la posición de la lista de Hijos de un ArbolGeneral que apunte a un Planeta Bot.

```
public bool Atacar(Planeta bot, Planeta objetivo)
{ //metodo auxiliar para determinar si el ataque se puede realizar con exito
    if(bot.population/2 > objetivo.population)
    {
        return true;
    }
    else
    {
        return false;
    }
}
2 referencias
public int Profundidad(ArbolGeneral<Planeta> arbol)
{
    if (arbol.getDatoRaiz().team == 2)
    {
        return 0;
    }
    foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
    {
        int profundidadEnHijo = Profundidad(hijo);
        if(profundidadEnHijo>=0)
        {
            return profundidadEnHijo+1;
        }
    }
    return -1;
}
```

Este método sirve para encapsular la forma en la que se decide si se ataca o no a un planeta. Sólo atacará cuando sea posible conquistarlo, y no atacara a planetas aliados (es decir, planetas conquistados por el Bot).

```
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol)
{
    bool ataque = false;

    Movimiento movimiento = null;
    if(arbol.getDatoRaiz().team == 2) //si el planeta es del bot
    {//esto solo sirve para atacar nodos hijos
        foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
        {
            if (hijo.getDatoRaiz().team != 2)
            {
                if(ataque = Atacar(arbol.getDatoRaiz(), hijo.getDatoRaiz()))
                {
                    return movimiento = new Movimiento(arbol.getDatoRaiz(),hijo.getDatoRaiz());
                }

            }
        }
    }
    else
    {
        foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
        {
            movimiento = CalcularMovimiento(hijo);
            if (movimiento != null)
            {
                break;
            }
        }
    }
    return movimiento;
}
```

La primer implementación del método de CalcularMovimiento fue esta. Se

partió desde la idea de que un nodo (planeta) sólo puede atacar a su nodo padre o a nodos hijos, por lo que primero se intentó con la forma en la que se podría atacar a un hijo. La siguiente captura mostrará la versión final de la implementación.

```
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol)
{
    bool ataque = false;

    Movimiento movimiento = null;
    if(arbol.getDatoRaiz().team == 2) //si el planeta es del bot
    {//esto solo sirve para atacar nodos hijos
        foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
        {
            if (hijo.getDatoRaiz().team != 2)//si el hijo no es bot, es objetivo
            {
                if(ataque = Atacar(arbol.getDatoRaiz(), hijo.getDatoRaiz()))
                {
                    return movimiento = new Movimiento(arbol.getDatoRaiz(),hijo.getDatoRaiz());
                }

            }
        }
    }
    if(arbol.getDatoRaiz().team != 2 && movimiento == null)
    {//si no pudo atacar al hijo, ataca al padre
        int posicion = ComprobarHijos(arbol);
        if (posicion >-1)
        {//si encontró un hijo bot, se evalúa la posibilidad de ataque
            if (Atacar(arbol.getHijos()[posicion].getDatoRaiz(), arbol.getDatoRaiz()))
            {//el hijo ataca al padre
                return movimiento = new Movimiento(arbol.getHijos()[posicion].getDatoRaiz(), arbol.getDatoRaiz());
            }
        }
    }
    if (movimiento == null)
    {//modificacion del else, porque sino no ejecuta esta parte del código
        //seguirá buscando un movimiento recursivamente
        foreach(ArbolGeneral<Planeta> hijo in arbol.getHijos())
        {
            movimiento = CalcularMovimiento(hijo);
            if (movimiento != null)
            {
                break;
            }
        }
    }
    return movimiento;
}
```

La versión final implementa una forma en la que puede atacar al nodo padre también. La idea en esencia fue “Si no puede atacar al hijo (Atacar = false), que intente atacar al padre”.

La siguiente captura es de un método auxiliar diseñado para poder implementar el método de Consulta 2.

```
public List<String> RecorridoPorNivelesLista(ArbolGeneral<Planeta> arbol)
{//el metodo solo es utilizado en planetas de bots, devolverá todos los descendientes más el nodo raiz
    List<String> descendientes = new List<String>();
    Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
    if(arbol!=null)
    {
        cola.encolar(arbol);

        while(!cola.esVacia()) //itera mientras la cola no esté vacía
        {
            int i = 1;
            ArbolGeneral<Planeta> nodoActual = cola.desencolar();
            descendientes.Add(i+")"+nodoActual.getDatoRaiz().population.ToString()+ ", ");
            i++;
            foreach(ArbolGeneral<Planeta> hijo in nodoActual.getHijos())
            {
                cola.encolar(hijo);
            }
        }
    }
    return descendientes;
}
```

La idea principal fue poder resolverlo con un recorrido por niveles, por lo que la lista devuelve eso. Se tuvo muchos problemas de casteo para poder lograr que al final, lo que se devolviese fuera un String, sin embargo fue solucionado con una sentencia que se implementa en el método de Consulta 2.

```
public String Consulta2( ArbolGeneral<Planeta> arbol)
{//devuelve un listado con todos los planetas descendientes del planeta bot
    List<String> listado = null;
    if(arbol.getDatoRaiz().EsPlanetaDeLaIA())
    {
        listado = RecorridoPorNivelesLista(arbol);
    }
    else
    {
        Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
        cola.encolar(arbol);
        while (!cola.esVacia())//recorre en post orden
        {
            ArbolGeneral<Planeta> nodoActual = cola.desencolar();
            foreach (ArbolGeneral<Planeta> hijo in arbol.getHijos())
            {
                if(hijo.getDatoRaiz().EsPlanetaDeLaIA())
                {
                    listado = RecorridoPorNivelesLista(hijo);
                    return string.Join(Environment.NewLine, listado);
                }
                else if (hijo != null)
                {

                    cola.encolar(hijo);
                }
            }
            while(!cola.esVacia())
            {
                string respuesta = Consulta2(cola.desencolar());
                if(respuesta != "")
                {
                    return respuesta;
                }
            }

        }
    }

    if(listado != null)
    {
        return string.Join(Environment.NewLine, listado);
    }
    else
    {
        return "";

    }

}
```

Esta es la implementación de la Consulta 2. Se tuvo muchos problemas tratando de capturar las excepciones que ocurrían por devolverse respuestas en blanco (String respuesta = “”;). El método terminó funcionando a partir de recorrido inorden. La idea es que solo se procese el nodo que sea de bot con el método auxiliar.

```
public String Consulta3( ArbolGeneral<Planeta> arbol)
{
    string resultado = "";
    Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
    Cola<ArbolGeneral<Planeta>> colaEnOrden = new Cola<ArbolGeneral<Planeta>>();

    cola.encolar(arbol);

    while (!cola.esVacia())//encolo en orden todos los nodos en una cola auxiliar
    {
        ArbolGeneral<Planeta> nodoActual = cola.desencolar();

        foreach (ArbolGeneral<Planeta> hijo in nodoActual.getHijos())
        {
            cola.encolar(hijo);
        }

        colaEnOrden.encolar(nodoActual);
    }

    ArbolGeneral<Planeta> raiz = colaEnOrden.desencolar();
    resultado += $"Nivel 1: Población Total: {raiz.getDatoRaiz().Poblacion()}, Población Promedio: {raiz.getDatoRaiz().Poblacion()}\n";
    int sumatoria = 0;
    for(int i = 1; i<=5;i++) //nivel 2
    {
        ArbolGeneral<Planeta> nodoActual = colaEnOrden.desencolar();
        sumatoria += nodoActual.getDatoRaiz().Poblacion();
    }
    double poblacionPromedio = sumatoria / 5;
    resultado += $"Nivel 2: Poblacion Total: {sumatoria}, Población Promedio: {poblacionPromedio:F2}\n";
    sumatoria = 0; //reseteo la sumatoria
    for(int i = 1; i<=15; i++) //empiezo las operaciones para nivel 3
    {
        ArbolGeneral<Planeta> nodoActual = colaEnOrden.desencolar();
        sumatoria += nodoActual.getDatoRaiz().Poblacion();
    }
    poblacionPromedio = sumatoria / 15;
    resultado += $"Nivel 3: Poblacion Total: {sumatoria}, Población Promedio: {poblacionPromedio:F2}\n";
    sumatoria = 0;
    for (int i = 1; i <= 15; i++) //empiezo las opeaciones para nivel 4
    {
        ArbolGeneral<Planeta> nodoActual = colaEnOrden.desencolar();
        sumatoria += nodoActual.getDatoRaiz().Poblacion();
    }
    poblacionPromedio = sumatoria / 15;
    resultado += $"Nivel 4: Poblacion Total: {sumatoria}, Población Promedio: {poblacionPromedio:F2}\n";
    return resultado;
}
```

Esta consulta sólo se pudo resolver de forma iterativa. Se cree que se puede resolver con muchas menos líneas de código con métodos recursivos, sin embargo no se halló la forma de llegar a eso.
El método tiene otra debilidad, sólo funciona con el tipo de árbol general que maneja el juego, ya que siempre genera la misma cantidad de nodos y la misma cantidad de niveles.

Observaciones:

Se trató de evitar usar clases que no fueran las dadas por la cátedra. Si la clase de ArbolGeneral tuviera una forma de poder consultar su altura, la implementación de los métodos pedidos por la consigna hubiera sido (posiblemente) más sencilla y menos costosa en términos de tiempo de ejecución.

Conclusiones:

El trabajo fue retador. Se comprobó la gran utilidad de poder usar métodos recursivos en el manejo de grafos (en este caso, en el manejo de Árbol General). Se pudo haber mejorado en varios aspectos el código, ya que quizás no fue lo ideal hacer tantos métodos auxiliares, sin embargo son de gran utilidad a la hora de tener que dividir un problema grande en varios más sencillos (paradigma de programación de divide y reinarás).
Una observación interesante es que al momento de ejecutar el código, no

puedo predecir qué es lo que va a hacer el bot. Hay sutilezas en la ejecución de CalcularMovimiento que hacen que no pueda saber bien a qué planeta va a atacar. Por último, sería interesante mejorar el método de Calcular Movimiento por uno que priorizara conquistar los planetas de mayor tamaño, o para transferir naves a planetas de bots (aunque eso no fue pedido por la consigna).