Complejidad Temporal, Estructuras de Datos y Algoritmos

TRABAJO PRÁCTICO FINAL

ALUMNO: ALAN LOPEZ

COMISIÓN: 5

ING. EN INFORMÁTICA

2DO CUATRIMESTRE

20/11/2023

ÍNDICE

INTRODUCCIÓN

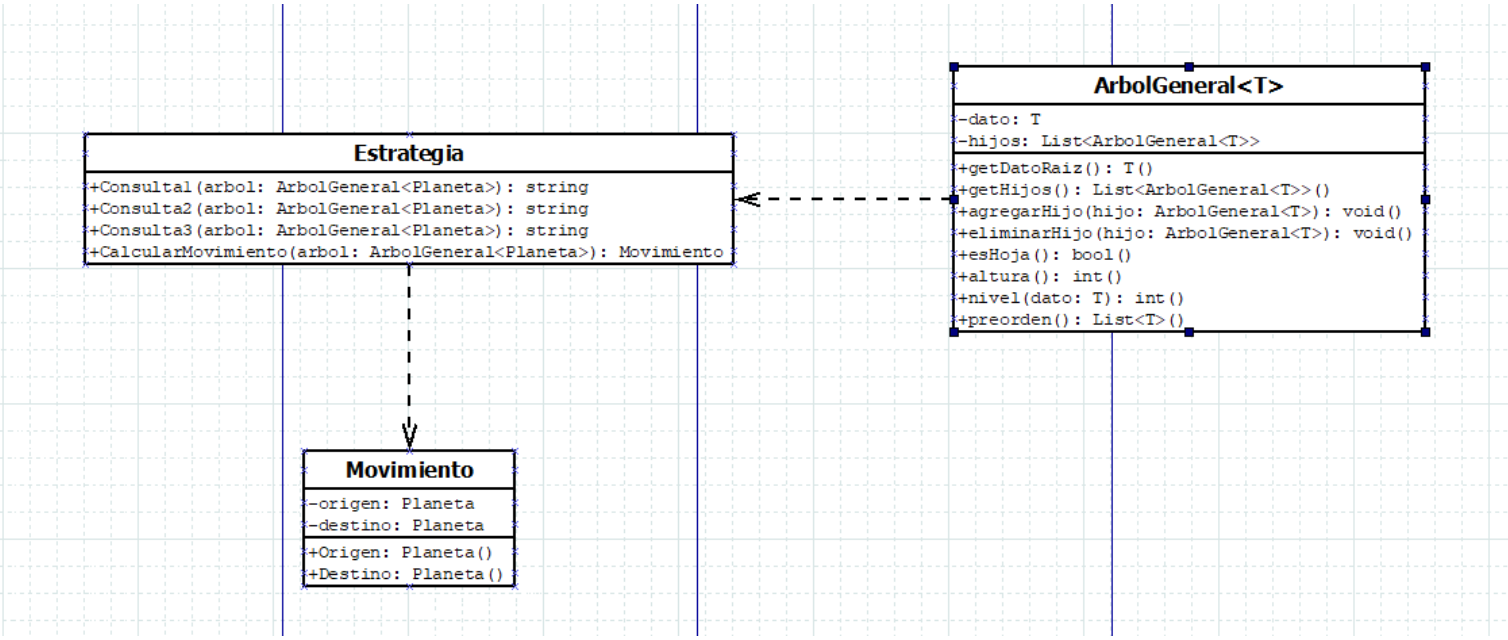
El presente informe se adentra en el universo de los árboles binarios, una estructura de datos fundamental que permite organizar y manipular información de manera jerárquica y eficiente. Exploraremos desde los conceptos básicos de cómo crear y representar árboles binarios hasta las técnicas de recorrido que nos permiten acceder y manipular sus nodos de forma ordenada.

En este caso, implementaremos el uso de los arboles binarios para la creación de estrategias y comportamientos de un juego en donde la inteligencia de la IA depende de cómo recorramos los distintos planetas/nodos según el recorrido del árbol que implementamos en el “universo” del juego.

Un excelente ejercicio muy visual e interactivo que nos evidencia los recorridos de los árboles y su implementación correspondiente en distintas ocasiones.

UML

Comenzamos planteando el UML donde se evidencian las relaciones entre las clases:



La clase estrategia depende de la clase árbol general porque requiere instancias de esta clase para poder realizar, por ejemplo: calcularMovimiento() utilizando los métodos de la clase ArbolGeneral para buscar los nodos del usuario y del bot en el árbol, buscar el camino más corto entre el usuario y el bot, y calcular la población del planeta más cercano al bot y del planeta más alejado del usuario.

Asi como también, Movimiento depende de Estrategia, porque puede devolver una instancia de la clase Movimiento. La clase Estrategia utiliza el método CalcularMovimiento() para calcular el movimiento que debe realizar el bot. El método CalcularMovimiento puede devolver una instancia de la clase Movimiento.

EJERCICIOS

Comenzamos con la **consulta 1**: Consulta1 (ArbolGeneral<Planeta> arbol): Calcula y retorna un texto con la distancia del camino que existe entre el planeta del Bot y la raíz.

```
1 referencia
public String Consulta1(ArbolGeneral<Planeta> arbol)
{
    List<Planeta> caminoBot = new List<Planeta>();

    bool BuscarCamino(ArbolGeneral<Planeta> arbolBusqueda, List<Planeta> caminoPlanetas)
    {
        if (arbolBusqueda != null)
        {
            caminoPlanetas.Add(arbolBusqueda.getDatoRaiz());

            if (arbolBusqueda.getDatoRaiz().EsPlanetaDeLaIA()) // si el bot esta en el planeta raiz
            {
                return true;
            }
        }

        foreach (ArbolGeneral<Planeta> arbolHijo in arbolBusqueda.getHijos()) //recursivamente itera y recorre el camino hasta dar con el bot
        {
            if (BuscarCamino((ArbolGeneral<Planeta>)arbolHijo, caminoPlanetas)) //si encuentra el bot retorna true
            {
                return true;
            }
        }

        caminoPlanetas.RemoveAt(caminoPlanetas.Count - 1);

        return false;
    }

    BuscarCamino(arbol, caminoBot);

    return $"CONSULTA 1:Distancia entre Planeta Central y ubicacion del bot: {caminoBot.Count - 1} planetas ";
}//distancia del camino que existe entre el planeta del Bot y la raiz. recursivo
```

Implementación consulta 1:

Lo primero que realizamos es crear la lista “caminoBot” que es donde almacenamos los planetas que forman parte del camino desde la raíz del árbol hasta el planeta donde se encuentra el bot.

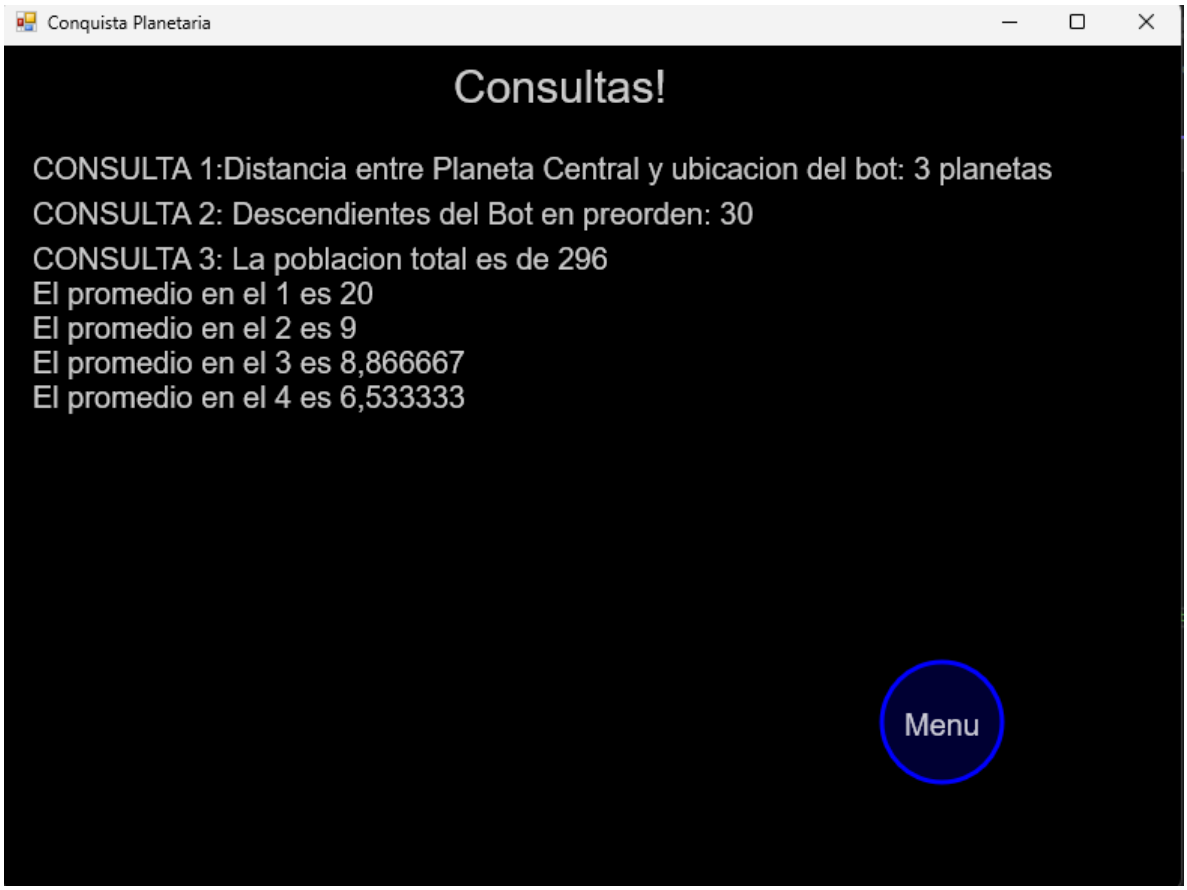
La función BuscarCamino implementa una búsqueda recursiva. Empieza desde la raíz del árbol y explora cada nodo en busca del planeta del bot. Agrega el planeta actual al camino (caminoBot) mientras recorre el árbol.

La búsqueda explora cada nodo del árbol. Al encontrar un nodo que corresponde al planeta donde está el bot, detiene la exploración y retorna true, indicando que se encontró el nodo del bot. Si no encuentra el bot en el nodo actual, avanza a los nodos hijos y continúa la búsqueda.

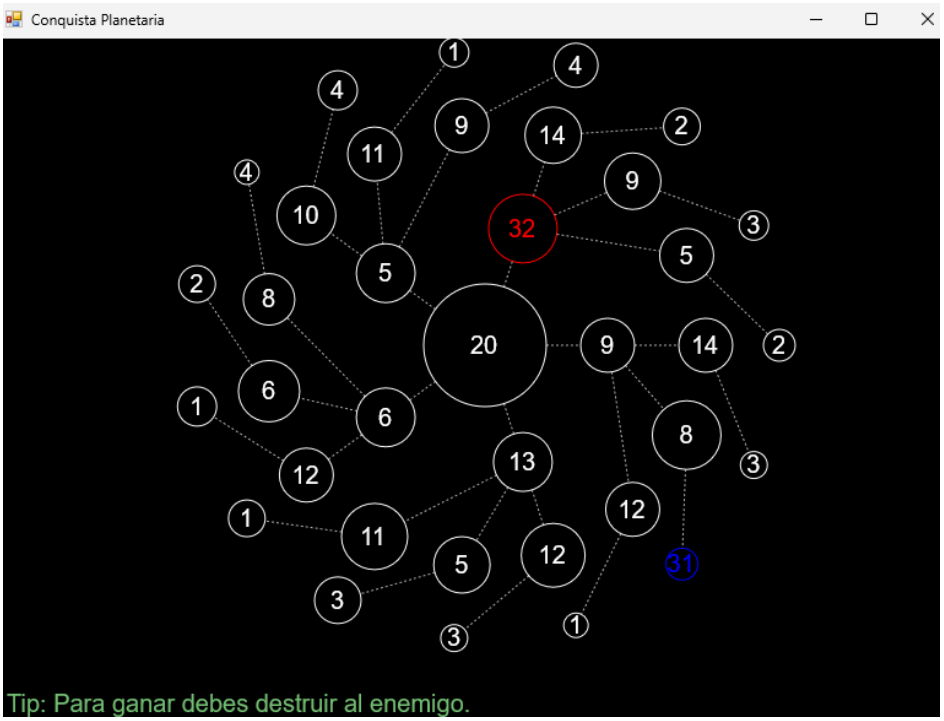
Durante la búsqueda, la lista caminoBot registra todos los planetas visitados desde la raíz hasta el planeta donde se encuentra el bot. Cuando la búsqueda finaliza, se retorna un mensaje que indica la distancia entre la raíz y el planeta del bot.

Esta distancia se calcula como la longitud del camino (caminoBot.Count - 1), representando la cantidad de planetas recorridos desde la raíz hasta el bot.

Ejecución del código consulta 1:

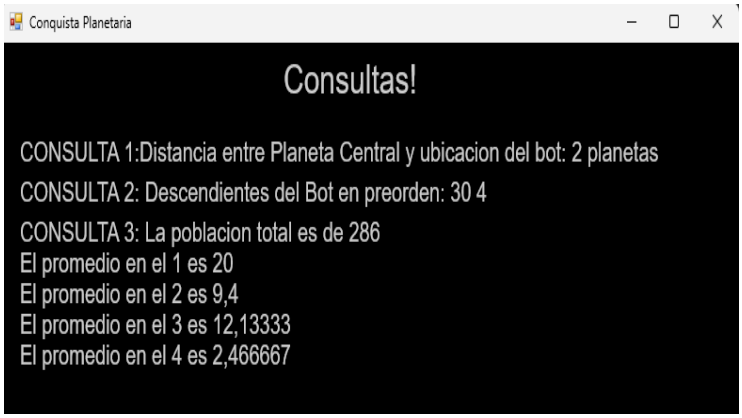


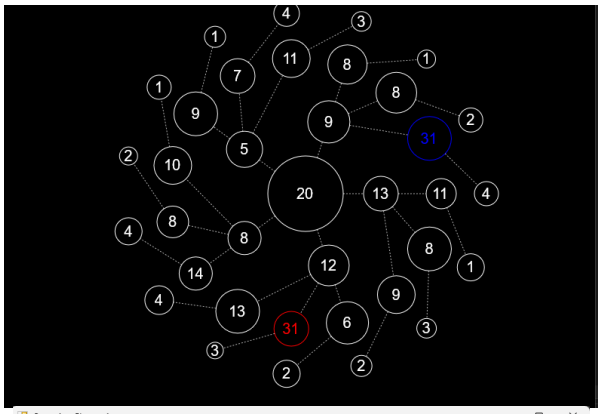
Observamos que la “consulta 1” indica una distancia entre el planeta central y el bot de 3 planetas. Verificamos dentro del juego:

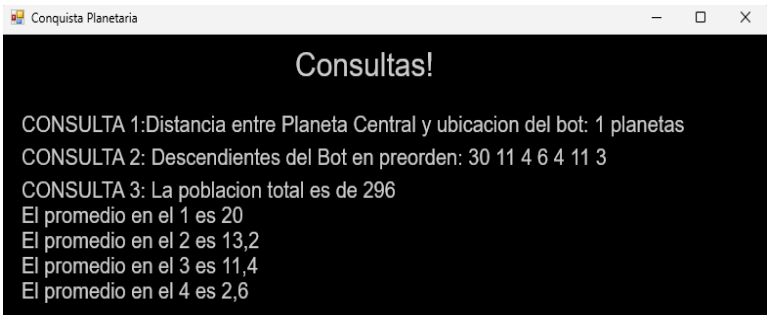


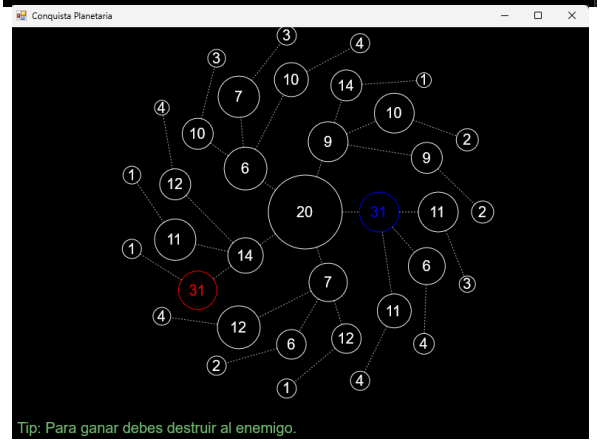
Efectivamente podemos verificar que el planeta azul del bot se encuentra a 3 planetas de la raíz.

A continuación, voy a brindar algunos ejemplos más:









Consulta 2: Consulta2 (ArbolGeneral<Planeta> arbol): Retorna un texto con el listado de los planetas ubicados en todos los descendientes del nodo que contiene al planeta del Bot.

```
1 referencia
public string Consulta2(ArbolGeneral<Planeta> arbol)
{
    List<Planeta> descendientesBot = new List<Planeta>(); //almacenamos los descendientes del bot

    void BuscarDescendientesBot(ArbolGeneral<Planeta> arbolBusqueda)
    {
        if (arbolBusqueda != null)
        {
            if (arbolBusqueda.getDatoRaiz().EsPlanetaDeLaIA()) //verificamos si la raiz es el bot
            {
                descendientesBot = arbolBusqueda.preorden(); // Obtiene todos los descendientes en preorden
            }
            else
            {
                foreach (ArbolGeneral<Planeta> hijo in arbolBusqueda.getHijos()) //recursivamente busca al bot
                {
                    BuscarDescendientesBot(hijo);
                }
            }
        }
    }

    BuscarDescendientesBot(arbol);

    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.Append("CONSULTA 2: Descendientes del Bot: "); // creamos la consulta y el formato como queremos mostrarlo
    foreach (Planeta planeta in descendientesBot)
    {
        stringBuilder.Append($"{planeta.Poblacion()} ");
    }
    return stringBuilder.ToString(); //lo retornamos como string
}//realiza un recorrido preorden retornando un string donde imprime primero raiz(bot)
                                                    //y descendientes(izquierdos primero y luego derechos)
```

```
2 referencias
public List<T> preorden()
{
    List<T> descendientesBot = new List<T>();

    descendientesBot.Add(this.dato); // Agrega el dato del nodo actual al recorrido

    foreach (ArbolGeneral<T> hijo in hijos)
    {
        List<T> descendientesHijo = hijo.preorden(); // Obtiene el recorrido preorden de cada hijo
        descendientesBot.AddRange(descendientesHijo); // Agrega los descendientes del hijo al recorrido
    }

    return descendientesBot;
}
```

Implementación consulta 2:

La función inicia explorando el árbol a partir del nodo raíz, Verifica si el nodo actual contiene al planeta del bot.

Si encuentra el nodo que contiene al bot, realiza un recorrido preorden de ArbolGeneral para obtener todos los descendientes de ese nodo.

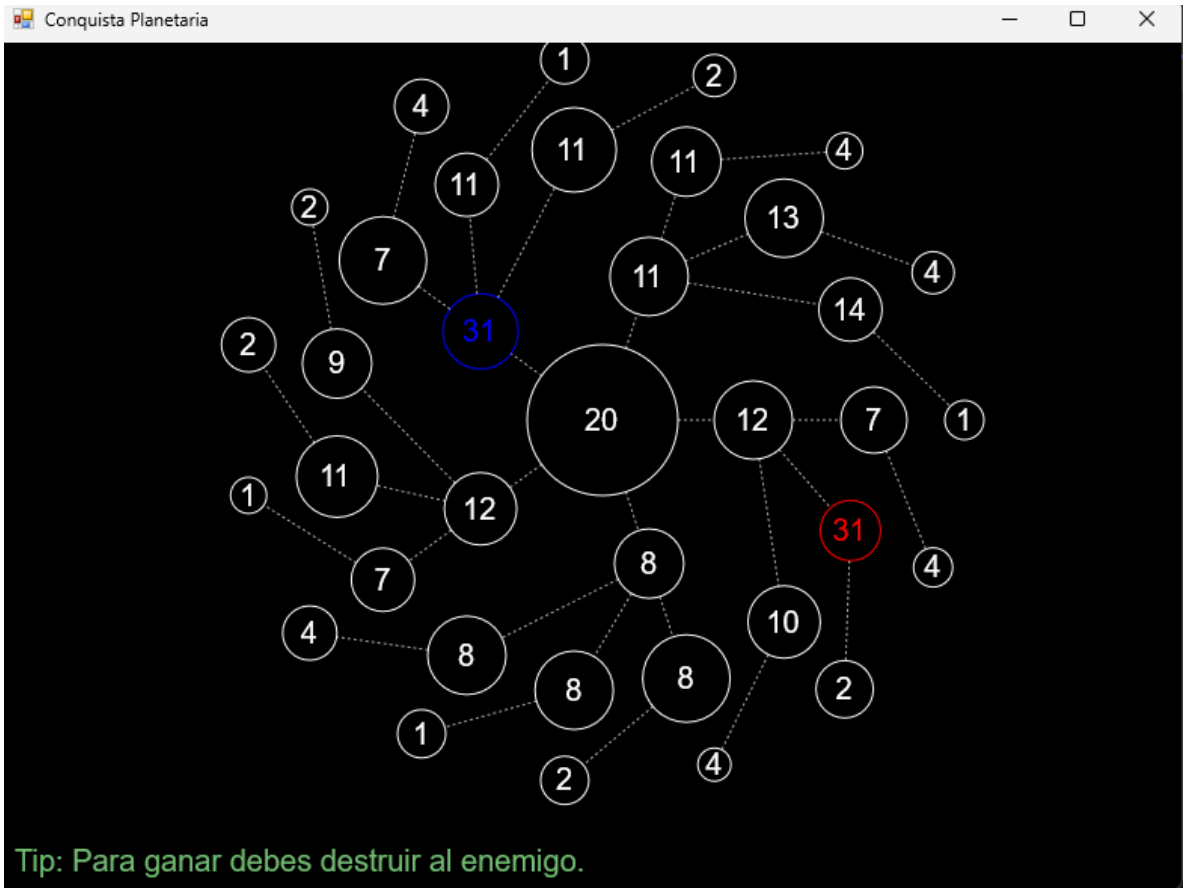
Este recorrido se efectúa explorando primero la raíz (el nodo del bot) y luego sus descendientes de izquierda a derecha.

Cuando obtenemos la lista con los descendientes al no tener nombre los planetas decidí indicar cuáles son sus descendientes según su población siendo esta el identificador de cada planeta.

Una vez que se ha recopilado la información de los descendientes del bot en el formato deseado, esta se devuelve como una cadena de texto.

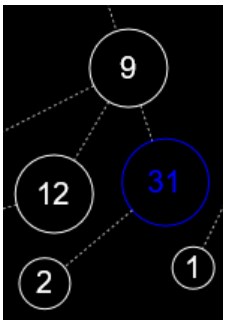
Ejecución del código consulta 2:

La consulta 2 nos indica que el nodo bot es el de población 30 y sus descendientes ordenados en preorden son 11,2,11,1,7 y 4. El número de cada uno corresponde a su población antes de iniciar el juego, ahora vemos como se cumple la consulta:



Tal como se puede observar el bot azul funciona como raíz de un árbol de los cuales se genera el recorrido correcto preorden retornando primero el 30 (en este caso es 31 por el pequeño delay entre ingresar al juego y hacer la captura) y sus descendientes de izquierda a derecha: 11,2,11,1,7 y 4.

Veamos cómo se ejecuta en otro ejemplo complementario:



En este caso la consulta dos nos indica simplemente que el bot (30) tiene un solo descendiente ósea su hijo (2).

Consulta 3: Consulta3 (ArbolGeneral<Planeta> arbol): Calcula y retorna en un texto la población total y promedio por cada nivel del árbol.

```
1 referencia
public String Consulta3(ArbolGeneral<Planeta> arbol)
{
    if (arbol == null) //verificamos que el arbol no este vacio
    {
        return "No hay planetas";
    }

    Cola<ArbolGeneral<Planeta>> colaAux = new Cola<ArbolGeneral<Planeta>>(); // creamos una cola para realizar un recorrido por niveles
    colaAux.encolar(arbol);
    colaAux.encolar(null); // Inicia encolando el nodo raiz (arbol) en la cola y coloca un marcador nulo (null) para distinguir niveles.

    int poblacionTotal = 0;
    int poblacionNivel = 0;
    int planetasPorNivel = 0;
    int niveles = 1;
    string texto = "";

    while (!colaAux.esVacia()) //Mientras la cola no esté vacía, se ejecuta
    {
        ArbolGeneral<Planeta> arbolAux = colaAux.desencolar();
        if (arbolAux != null)//Si no es nulo, calcula la población total acumulando la población de cada planeta
        {
            poblacionTotal += arbolAux.getDatoRaiz().Poblacion();
            poblacionNivel += arbolAux.getDatoRaiz().Poblacion();
            planetasPorNivel++;
            if (arbolAux.getHijos().Count > 0) //Si tiene hijos, los encola para ser procesados en el siguiente nivel.
            {
                foreach (ArbolGeneral<Planeta> arbolHijo in arbolAux.getHijos())
                {
                    colaAux.encolar(arbolHijo);
                }
            }
        }
        else //Si encuentra un marcador nulo, se completo el nivel
        {
            texto += $"El promedio en el {niveles} es {(float)poblacionNivel / planetasPorNivel}\n"; //Calcula el promedio de población para el nivel actual
            poblacionNivel = 0;
            planetasPorNivel = 0; //Reinicia las variables para el siguiente nivel
            niveles++;
            if (!colaAux.esVacia())
            {
                colaAux.encolar(null); //avanza al siguiente nivel
            }
        }

    }

    return $"CONSULTA 3: La poblacion total es de {poblacionTotal}\n{texto}";// Retorna la población total del sistema de planetas y promedios de población por nivel.
}//población total y promedio por cada nivel del árbol.
```

Implementación consulta 3:

Verificamos si el árbol está vacío. Si lo está, devuelve un mensaje indicando la ausencia de planetas en el sistema.

Utiliza una cola (colaAux) para realizar un recorrido por niveles en el árbol. Inicia encolando el nodo raíz (arbol) en la cola y coloca un marcador null para distinguir niveles.

Mientras la cola no esté vacía, se ejecuta un bucle. Desencola un nodo (arbolAux) de la cola y verifica si es nulo: Si no es nulo, calcula la población total acumulando la población de cada planeta y calcula la población y cantidad de planetas en el nivel actual.

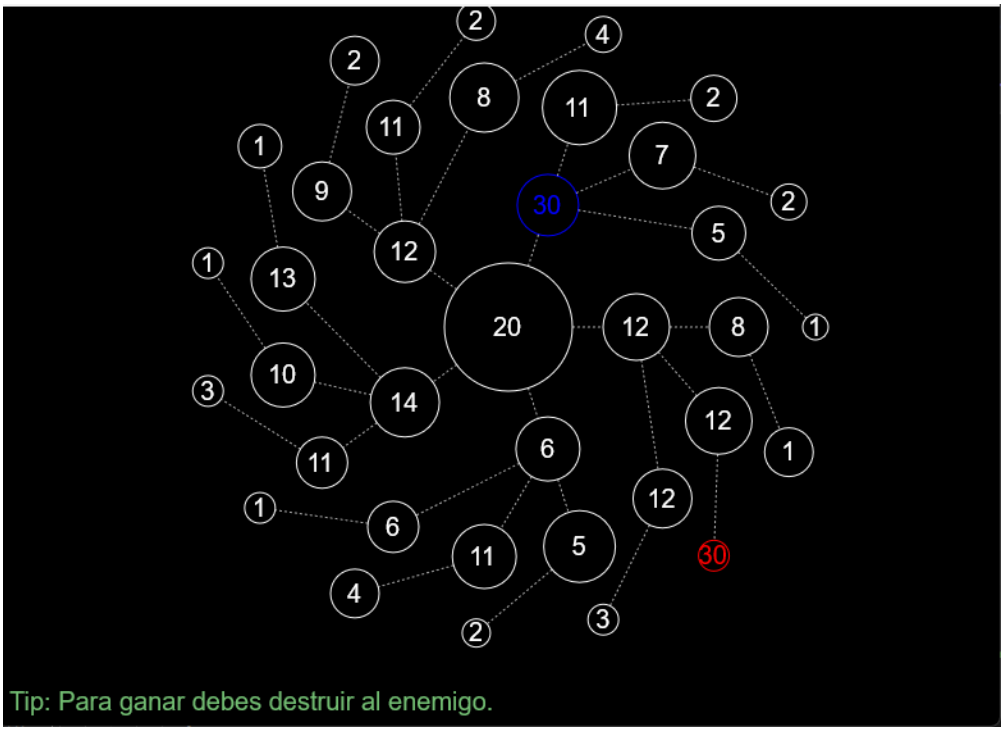
Si tiene hijos, los encola para ser procesados en el siguiente nivel.

Si encuentra un marcador nulo, se ha completado un nivel: Calcula el promedio de población para el nivel actual y lo agrega al texto de salida.

Reinicia las variables para el siguiente nivel y avanza al siguiente nivel.

Muestra la población total al final y cada uno de los promedios de población por nivel. Retorna un texto que resume la población total del sistema de planetas y los promedios de población por nivel.

Ejecución del código consulta 3:



Una vez ejecutado el código podemos observar que la población total presume ser de 292.

Nivel 1=20 Nivel 2=74 Nivel 3=139 Nivel 4=59

20+74+139+59=292. El resultado es correcto.

Luego hace los promedios:

Nivel 1= 20/1=20 Nivel 2=74/5=14.8 Nivel 3=139/15=2.6

Nivel 4= 59/15=3.93

Los resultados de los promedios son correctos.

Calcular 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 arbol de tipo ArbolGeneral<Planeta>. Cada nodo del árbol contiene como dato un planeta del juego.

```
1 referencia
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol)
{

    // Buscar nodo del usuario
    List<Planeta> caminoUser = new List<Planeta>();


    // Buscar nodo del bot
    List<Planeta> caminoBot = new List<Planeta>();


    bool BuscarCamino(ArbolGeneral<Planeta> arbolBusqueda, List<Planeta> caminoPlanetas, string player) //función recursiva para encontrar los nodos del usuario y del bo
    {
        if (arbolBusqueda != null)
        {
            caminoPlanetas.Add(arbolBusqueda.getDatoRaiz());

            if (arbolBusqueda.getDatoRaiz().EsPlanetaDeLaIA() && player == "bot")  //la busqueda se detiene cuando se encuentra al bot o cuando se encuenta al jugador
            {
                return true;
            }
            if (arbolBusqueda.getDatoRaiz().EsPlanetaDelJugador() && player == "user")
            {
                return true;
            }
        }


        foreach (ArbolGeneral<Planeta> arbolHijo in arbolBusqueda.getHijos())
        {
            if (BuscarCamino((ArbolGeneral<Planeta>)arbolHijo, caminoPlanetas, player))
            {
                return true;
            }
        }

        caminoPlanetas.RemoveAt(caminoPlanetas.Count - 1);

        return false;
    }

    BuscarCamino(arbol, caminoBot, "bot");
    BuscarCamino(arbol, caminoUser, "user");

    // Aplicamos una busqueda que se corte cuando encontramos ambos nodos

    Planeta planetaCercanoBot = null;
    Planeta planetaDestino = null;
```

```
        foreach (Planeta planeta in caminoUser) //buscamos los planetas especificos entre el bot y el usuario
        {
            if (planeta.EsPlanetaDeLaIA())
            {
                planetaCercanoBot = planeta;
            }
            if (planetaCercanoBot != null && planetaDestino == null && !planeta.EsPlanetaDeLaIA())
            {
                planetaDestino = planeta;
            }
        }

        if (planetaCercanoBot == null) // por si no se encontró ningún planeta que pertenezca al bot se busca el ultimo camino conocido
        {
            planetaCercanoBot = caminoBot[caminoBot.Count - 1];
            planetaDestino = caminoBot[caminoBot.Count - 2];
        }

        if (planetaCercanoBot.Poblacion() < planetaDestino.Poblacion() + 5)  // aca se identifica el camino al user con el planeta mas grande en poblacion y la diferencia que debe tener
                                                                             //+ de 5 de la poblacion que va a enviar
        {

            List<Planeta> caminoHumanoBot = new List<Planeta>(caminoUser);
            caminoHumanoBot.Reverse(); //invierte la lista del camino
            caminoHumanoBot.Remove(caminoHumanoBot[caminoHumanoBot.Count - 1]); //elimina el ultimo elemento
            caminoHumanoBot.AddRange(caminoBot); //agrega los elementos de caminobot al final de caminoHumano bot

            void AgregarNodosBots(ArbolGeneral<Planeta> estado, List<Planeta> lista) //busca los nodos planetas del bot y los agrega a la lista como argumento de forma recursiva
            {                                                                         //explora cada rama

                if (estado.getDatoRaiz() != null && estado.getDatoRaiz().EsPlanetaDeLaIA() && !lista.Contains(estado.getDatoRaiz()))
                {
                    lista.Add(estado.getDatoRaiz());
                }

                foreach (ArbolGeneral<Planeta> arbolHijo in estado.getHijos())
                {
                    AgregarNodosBots(arbolHijo, lista);
                }
            }

            AgregarNodosBots(arbol, caminoHumanoBot); //busca los planetas del bot y los agrega

            Planeta planetaBotMaxPoblacion = null;
            int indexCamino = -1;

            for (int i = 0; i < caminoHumanoBot.Count; i++) //busca la población más alta entre los controlados por bot en caminoHumanoBot y lo asigna a planetaBotMaxPoblacion
                                                            // actualiza indexCamino con la posición de ese planeta en la lista
            {
                if (caminoHumanoBot[i].EsPlanetaDeLaIA() && planetaBotMaxPoblacion == null)
                {
                    planetaBotMaxPoblacion = caminoHumanoBot[i];
                    indexCamino = i;
                }
                if (caminoHumanoBot[i].EsPlanetaDeLaIA() && planetaBotMaxPoblacion != null && planetaBotMaxPoblacion.Poblacion() < caminoHumanoBot[i].Poblacion())
                {
                    planetaBotMaxPoblacion = caminoHumanoBot[i];
                    indexCamino = i;
                }
            }

            Planeta destinoPoblacion = caminoHumanoBot[indexCamino - 1];

            return new Movimiento(planetaBotMaxPoblacion, destinoPoblacion); //realiza el movimiento
        }

        return new Movimiento(planetaCercanoBot, planetaDestino); //realiza el movimiento A
    }
}
```

bool List<Planeta>.Contains(Planeta item) (+ 2 sobrecargas)
Determina si un elemento se encuentra en List<T>.

Devuelve:
true si item se encuentra en la matriz List<T>; en caso contrario, false.

(variable local) Planeta planetaBotMaxPoblacion

Implementación calcular movimiento:

Primero creamos las listas donde se almacenan los caminos tanto al bot como al usuario.

Utiliza una función recursiva BuscarCamino para encontrar los nodos del usuario y del bot en el árbol. Realiza una búsqueda recursiva en arbol para encontrar el nodo del usuario y del bot. Añade cada nodo visitado al camino correspondiente (usuario o bot).

La búsqueda se detiene cuando encuentra el nodo del bot o del usuario.

Una vez encontrados los nodos del usuario y del bot, busca el nodo más cercano del bot y un destino.

Si no se encuentra un planeta más cercano al bot, establece el nodo final del camino del bot como el más cercano y el segundo nodo del final como el destino (planetaDestino).

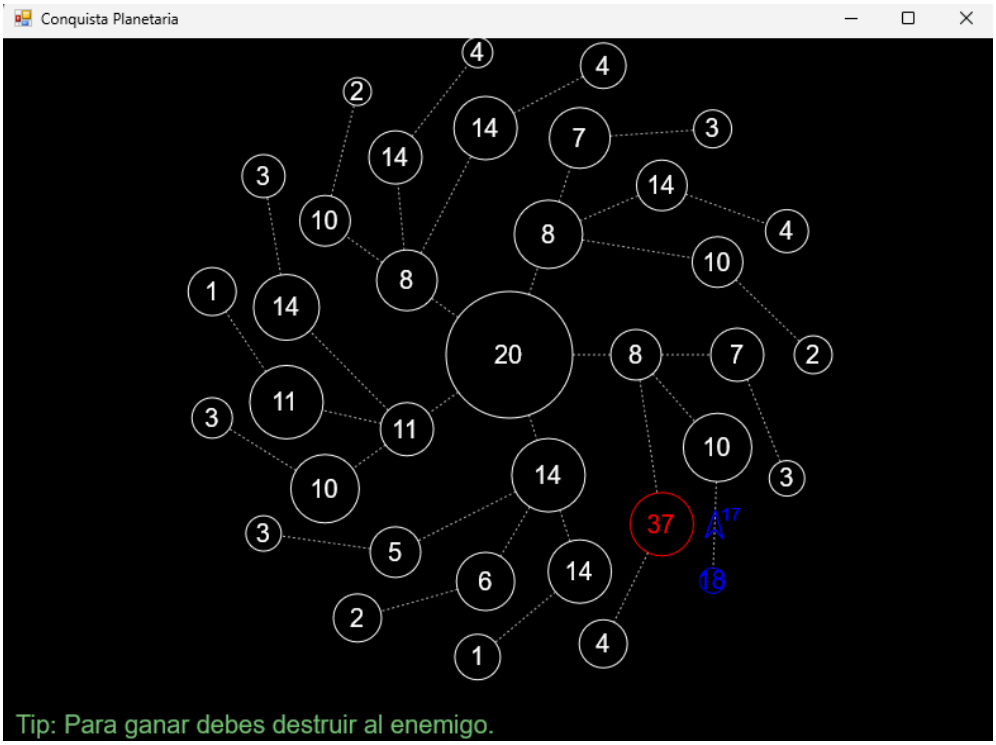
Verifica si la población del planeta más cercano al bot (planetaCercanoBot) es menor que la suma de la población del destino +5. Si se cumple la condición, se ejecuta para encontrar un nuevo destino y un nuevo movimiento.

Crea un nuevo camino combinando los caminos del usuario y del bot para encontrar un nuevo destino. Este nuevo camino (caminoHumanoBot) se compone de los nodos del usuario hasta el punto más cercano al bot y luego los nodos del bot.

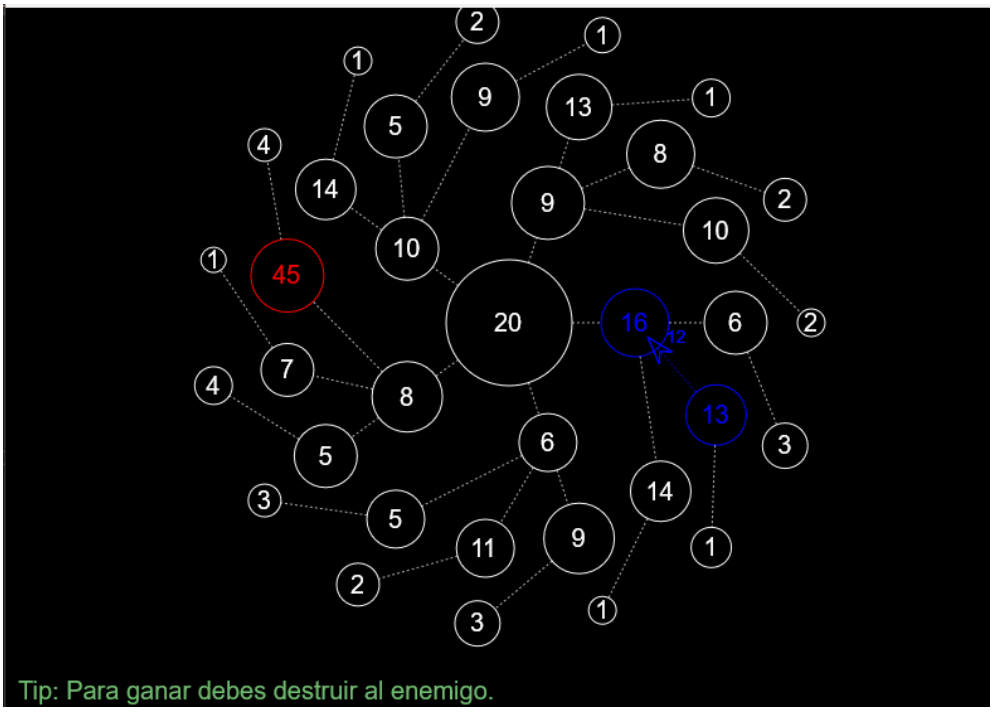
Se crea una función AgregarNodosBots que agrega los nodos del bot al nuevo camino creado. Se busca en el nuevo camino (caminoHumanoBot) el nodo del bot con la mayor población y se define como planetaBotMaxPoblacion.

Se elige el nodo anterior al nodo con la mayor población del bot como el destino (destinoPoblacion). Devuelve un objeto Movimiento que representa la mejor decisión de movimiento teniendo en cuenta las condiciones y el estado del juego.

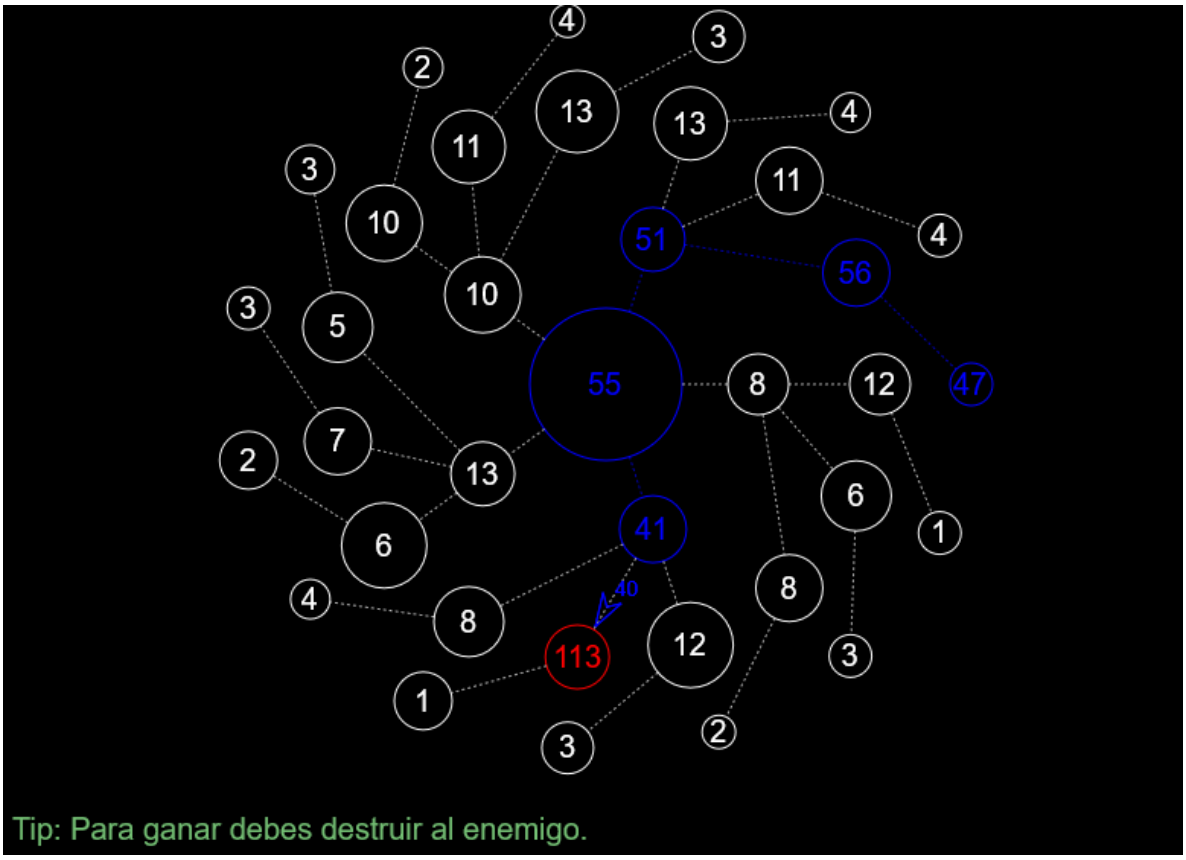
Ejecución del código calcular movimiento:



En este caso como el planeta origen es mayor que el planeta destino el bot envía las tropas para tomarlo.



En este otro, como el origen es mayor de 5 en diferencia decide enviar tropas de apoyo como estrategia.



En este ejemplo: si la población del planeta controlado por el bot es menor que la población del planeta objetivo del usuario más un margen de 5 unidades, el bot decide debilitar al planeta del usuario.

IDEAS/SUGERENCIAS

Luego de la implementación de los conocimientos adquiridos para la realización del proyecto surgieron algunas ideas tales como la implementación de un sistema de dificultades el cual permita seleccionar que tan difícil será la batalla contra el bot, además de por ejemplo, un sistema de habilidades especiales a la hora de tomar un planeta del enemigo el cual nos permita por ejemplo congelar ciertos nodos/planetas por un tiempo limitado tanto para colonizarlo o como para que no pueda enviar tropas si pertenece al rival, o alguna otra habilidad que permita sumar a nuestra elección las tropas de dos planetas neutros convirtiéndolos en un único planeta con la suma de ambas poblaciones.

A su vez, a modo de facilitar simplemente la actividad a realizar sin pensar tanto en la jugabilidad cuando entramos previamente a iniciar el juego, al apartado de consultas, sería correcto y muy beneficioso obtener una vista previa del juego dentro de ese apartado así cuando reposicionamos para realizar las pruebas no es necesario ejecutar el juego para ver las posiciones iniciales.

CONCLUSIÓN

En lo personal, disfrute mucho la realización del trabajo. Principalmente porque podía sentir que veía de forma más grafica el desafío a implementar y cada uno de mis avances en un trabajo que se asemeja un poco más a lo que es un trabajo real.

El principal desafío lo encontré en el desarrollo de calcular movimiento, llevo mucho tiempo desarrollar la estrategia, entender cómo se movería el bot y a partir de que método encontraría la forma de decidir entre los planetas neutrales.

Este trabajo ayudo a verificar las bases del conocimiento adquirido durante la cursada, a establecer un compromiso con el tiempo frente a la pantalla ante un trabajo desafiante y, según mi opinión, extenso que permitió también obligadamente desarrollar una rutina para el correcto desarrollo del mismo.