INSTITUTO INGENIERÍA Y AGRONOMÍA | Carrera Ingeniería en Informática

Complejidad Temporal, Estructuras de Datos y Algoritmos

Trabajo Práctico Final

Alumno: Espinosa Emanuel Roberto

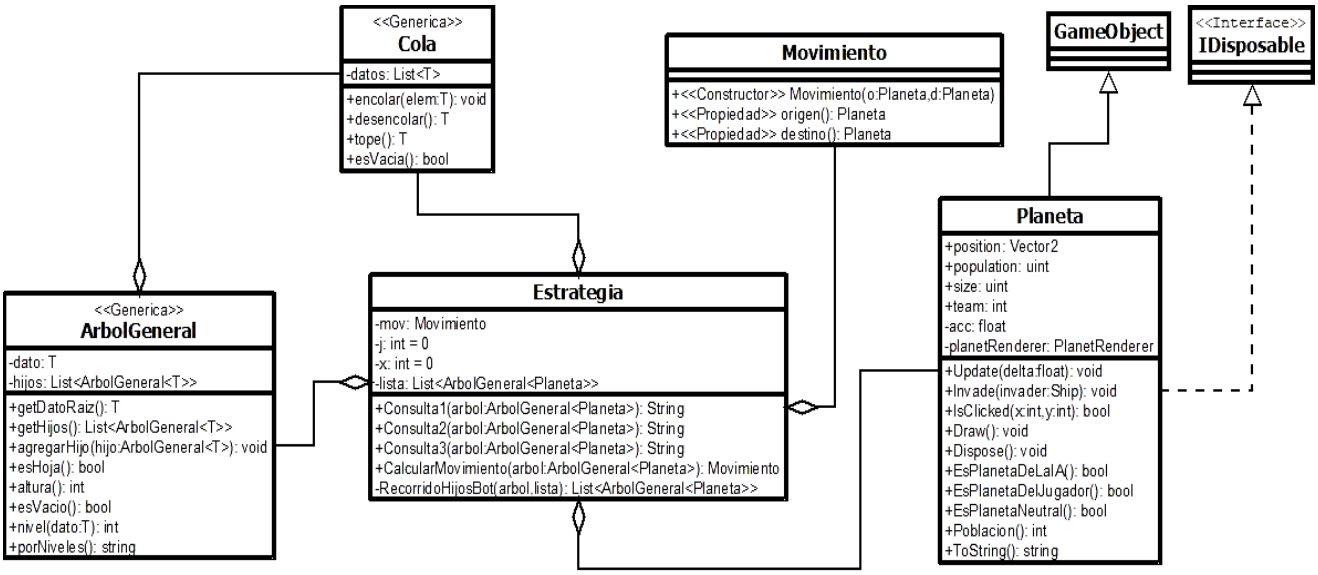
Docente: Ing. Alejandro Fontan

Comisión: 4

Introducción

El presente informe tiene como principal objetivo mostrar el desarrollo del Trabajo Práctico Final de la materia Complejidad Temporal, Estructura de Datos y Algoritmos. Se procedió a codificar y desarrollar la inteligencia artificial de un juego basado en una conquista planetaria donde el enfrentamiento se da únicamente entre dos jugadores.

Estructura y Metodología

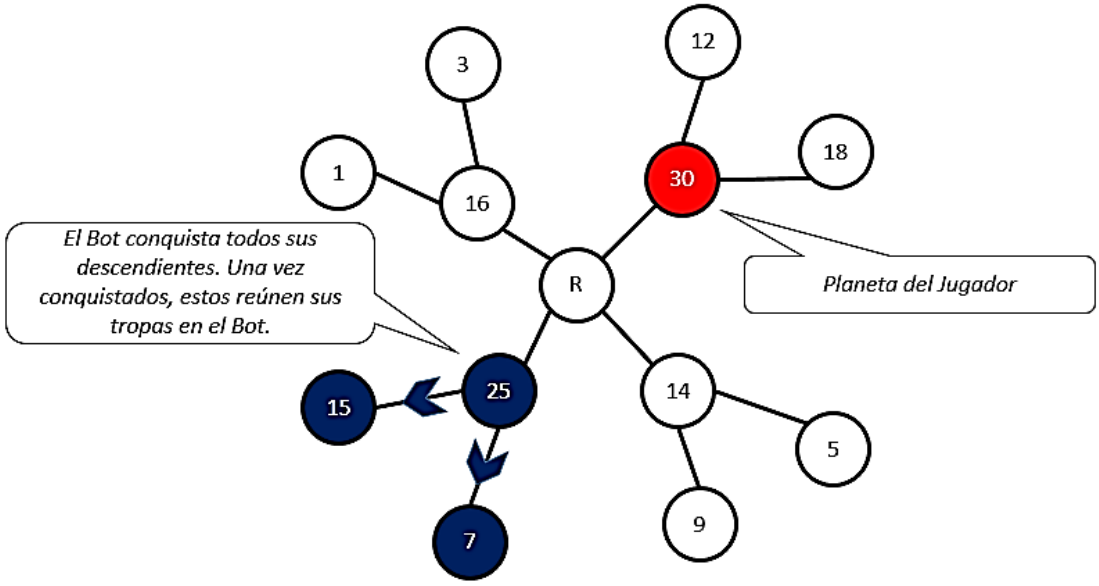


Se desarrollaron las consultas 1, 2 y 3 con un recorrido por niveles, utilizando una Cola y un separador por nivel (null). Para las tres consultas, la metodología implementada fue la misma. A continuación, se detalla el objetivo de respuesta de cada una de las consultas:

- **Consulta 1:** Calcula y retorna un texto con la distancia del camino que existe entre el planeta del Bot y la raíz.
- **Consulta 2:** Retorna un texto con el listado de los planetas ubicados en todos los descendientes del nodo que contiene al planeta del Bot.
- **Consulta 3:** Calcula y retorna en un texto la población total y promedio por cada nivel del árbol.
  - Se calcula la población total por cada nivel del árbol.
  - Una vez obtenida la población total por cada nivel, se calcula el promedio de población del árbol.
  - Este método retorna la población por cada nivel, la población total del árbol y el promedio de población del mismo.
  - El promedio se calcula como: $\frac{suma\ de\ cantidad\ de\ nodos\ por\ nivel}{cantidad\ de\ niveles}$

Por otra parte, la **Estrategia** desarrollada tiene como objetivo:

- Detectar el planeta del Bot
- Conquistar todos los planetas descendientes de éste.
- Reagrupar en el Bot las naves producidas en los planetas conquistados.



Código C#

Consulta 1:

```
public String Consulta1(ArbolGeneral<Planeta> arbol)
{
   /*Se supone que el árbol pasado como parámetro corresponde al nodo raíz*/
   Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
   ArbolGeneral<Planeta> arbolAux;
   int contNiv = 0, nivel = 0;
   c.encolar(arbol);     //Se encola el árbol
   c.encolar(null);      //Se encola el separador (null)
   while (!c.esVacia())  //Se ejecutara mientras la cola no este vacía
   {
      arbolAux = c.desencolar();
      if (arbolAux == null)
      {
          contNiv++;  //Si arbolAux es null, se incrementa el contador de niveles.
          if (!c.esVacia())
              c.encolar(null);  //Si quedan elementos en la cola, se encola el separador.
      }
       //Si el this actual es el Bot, entonces nivel es igual a contNiv
      if (arbolAux != null && arbolAux.getDatoRaiz().EsPlanetaDeLaIA())
          nivel = contNiv;
```

```
      else
      {
         //Si arbolAux no es null, no es el Bot y tiene hijos, entonces, Se recorren los hijos
         de arbolAux y se encolan.
         if (arbolAux != null && arbolAux.getHijos().Count != 0)
         {
             foreach (var hijo in arbolAux.getHijos())
                   c.encolar(hijo);
         }
      }
   }
   return $"* Existe una distancia de < {nivel.ToString()} > entre el planeta del Bot y la raíz";
}
```

Consulta 2:

```
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(); //Se invoca al método porNiveles de ArbolGeneral
          break; //Como ya se encontró el Bot se detiene la ejecución del bucle
      }
      else
      {
          if (arbolAux != null && arbolAux.getHijos().Count != 0)
          {
              foreach (var hijo in arbolAux.getHijos())
                    c.encolar(hijo);
          }
      }
   }
   return "\n* El planeta del Bot es: " + "["+arbolAux.getDatoRaiz()+"]" + "\n" +
            "  Los descendientes del Bot son: " + cadena;
}
```

Consulta 3:

```
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);
       }
    }
    double prom = contNodosNivel / contNiv;
    return cadena += "\n  Población total del árbol: " + contNodosNivel + "\n"
                   + "  Promedio de población por nivel: " + prom;
}
```

Estrategia:

```
private static Movimiento mov;
private static int j = 0, x = 0;
private static List<ArbolGeneral<Planeta>> lista = new List<ArbolGeneral<Planeta>>();


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);
         }
     }
     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)
{
    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;
}
```

Estrategia

Como se puede observar, para el desarrollo de la estrategia, se implementaron 2 métodos:

- **CalcularMovimiento** es el método encargado de retornar los movimientos necesarios para llevar a cabo el cumplimiento de la estrategia solicitada. Recibe por parámetro un ArbolGeneral<Planeta>, siendo este el nodo Raíz del arbol, que luego es añadido o encolado en una cola de tipo ArbolGeneral<Planeta>.
  Se utiliza un bucle while que va a ejecutarse hasta que la cola se vacíe o hasta que se encuentre el planeta de la IA. De ocurrir la segunda opción, entonces se realizan las operaciones necesarias. Luego con fines de optimización del algoritmo, se detiene dicho bucle con la instrucción break, ya que, si se encontró lo buscado, no es necesario que siga recorriendo la colección.
  La variable arbolAux servirá, como su nombre lo indica, como un auxiliar donde se guardará la referencia de cada elemento de la cola, conforme se vayan desencolando en cada iteración del bucle. Cuando la comparación de arbolAux con el planeta de la IA resulte verdadera, se invocará al método RecorridoHijosBot, siendo este el método que devolverá la lista de hijos del planeta de la IA.
  Una vez obtenida la lista de descendientes del Bot, se recorre con un bucle while y con la ayuda de una variable estática ***j*** de tipo *int* que actuará como índice. Este bucle se ejecutará mientras el valor del índice sea menor a la longitud de la lista - 1.
  En este nuevo recorrido se establece que si la posición [*j + 1*] no es un planeta de la IA y si esta misma posición se corresponde con una hoja, entonces se realiza el movimiento, desde la posición [*j*] a la posición [*j + 1*], de manera que el planeta ubicado en la posición [*j + 1*] sea conquistado. Luego se incrementa el índice ***j*** en una unidad y se retorna el movimiento.
  En caso de no cumplirse la condición recién mencionada, el movimiento se realiza desde la posición [*0*] (posición de la raíz, en este caso la posición del Bot) a la posición [*j + 1*] y al igual que en el caso anterior se incrementa el índice en una unidad y se retorna el movimiento.
  En caso que la condición del principio no se cumpla (arbolAux es planeta de la IA), se procede a comprobar si arbolAux tiene hijos, de ser así, se recorre la lista de hijos de este y se encolan en la cola mencionada anteriormente, para luego repetirse con cada uno el procedimiento descripto.
  Para que los planetas conquistados, envíen tropas al Bot, se utiliza una variable ***t*** de tipo *int* que almacenara la cantidad de elementos que tiene la lista de descendientes del Bot, a esta se le restara una unidad y una variable estática ***x*** que se incrementara cada vez que se procese un elemento de dicha lista. Para dicho procesamiento, se utiliza nuevamente un bucle while que se ejecutara hasta que la variable ***t*** sea mayor que 0. Dentro de este bucle se comprueba si la posición ***t*** de la lista es

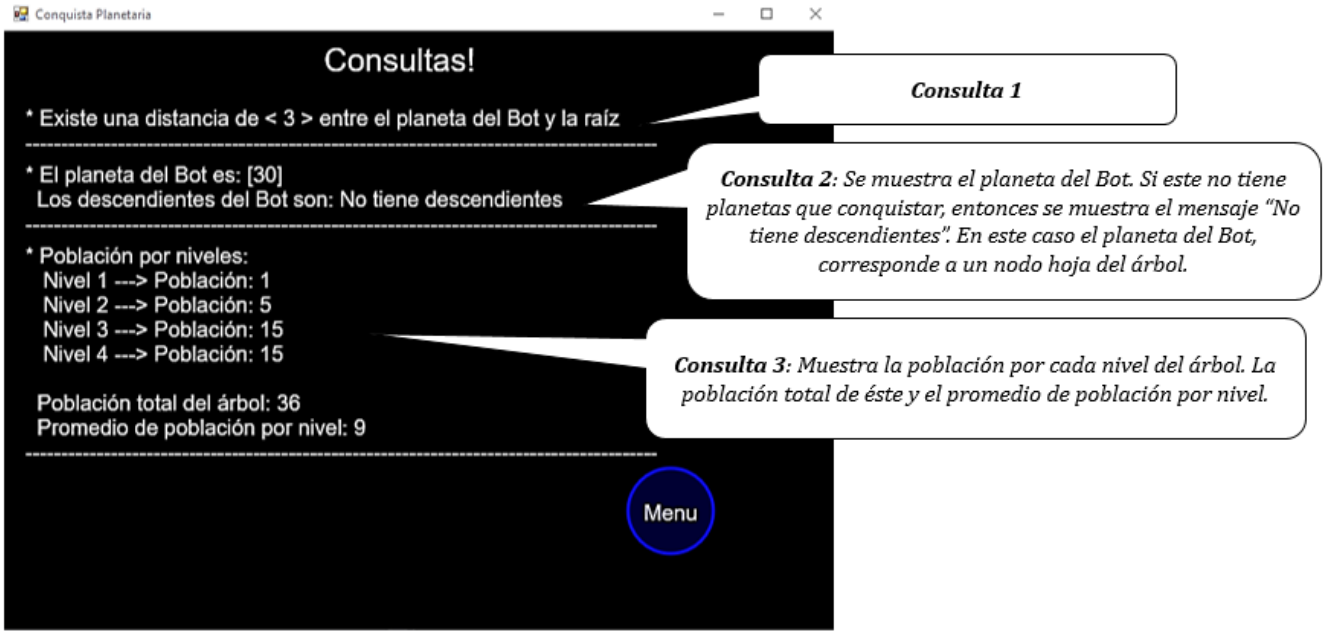
una hoja, si esto es afirmativo entonces se realiza un movimiento desde la posición [*t*] a la posición [*t – 1*], se incrementa la variable ***x***, y se retorna el movimiento.
En caso de no cumplirse la condición el movimiento será de la posición [*t*] a la posición [*0*] y al igual que en el caso anterior se incrementa la variable ***x***, y se retorna el movimiento. De esta forma, cada uno de los planetas conquistados, reagruparan sus tropas en el planeta de la IA.

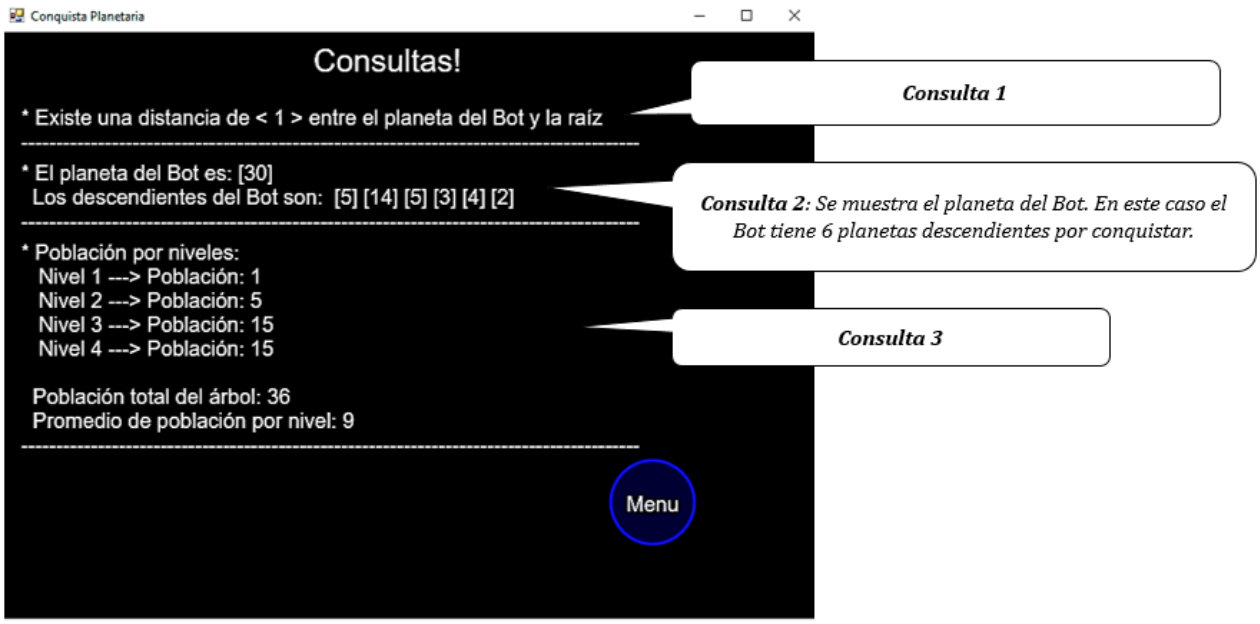
- **RecorridoHijosBot**, es un método recursivo que recibe como parámetro un árbol y una lista donde se guardarán los hijos de este. Dentro del método se utiliza una lista auxiliar que servirá de ayuda para el correcto funcionamiento del mismo. Se agrega el arbol recibido, a la lista pasada como parámetro, si este arbol es una hoja, entonces se retorna la lista (caso Base).
Si el árbol no es una hoja, entonces se recorre la lista de hijos del arbol, dentro de este recorrido, se utiliza la lista auxiliar para invocar recursivamente al método, pero esta vez pasándole como arbol, el elemento actual de la iteración. Luego, se verifica si la lista auxiliar es diferente de null, de ser así se retorna esta lista auxiliar.

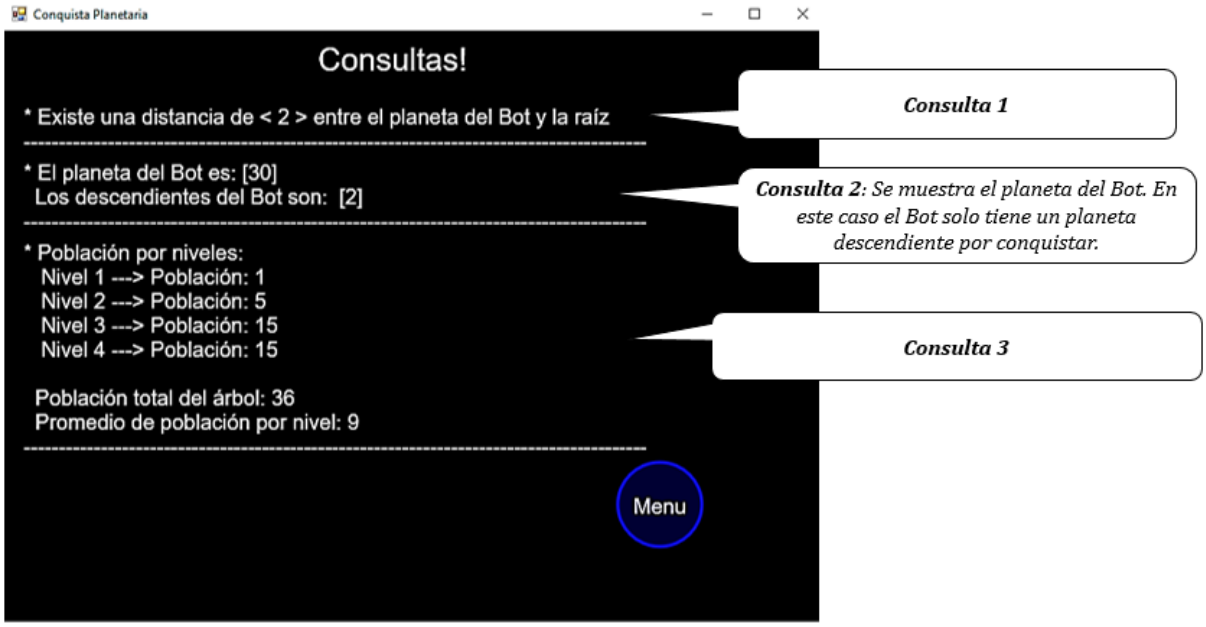
Demostración de Funcionamiento

- Consultas

A continuación, se muestran las posibles respuestas que puede arrojar las consultas:

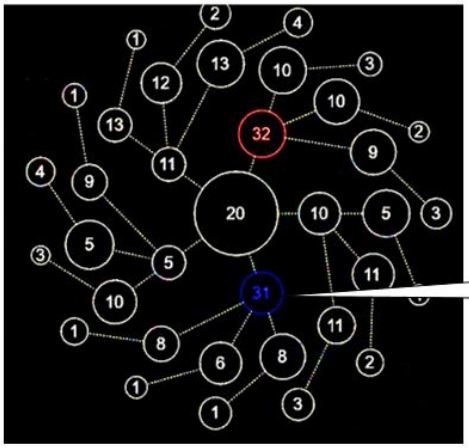






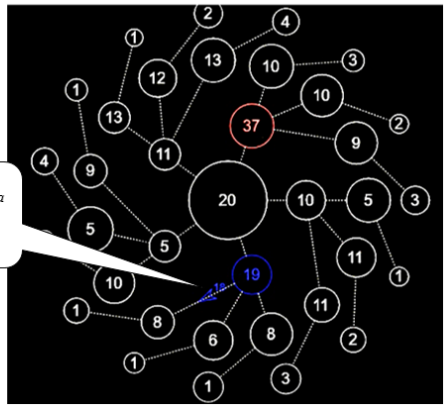
- Estrategia

  En las siguientes imágenes se muestra como actúa la estrategia desarrollada en el presente trabajo, al iniciar el juego.

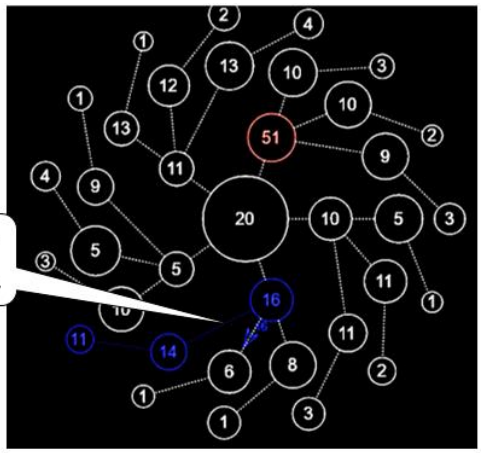


Planeta del Bot. *Va a conquistar a todos sus descendientes.*

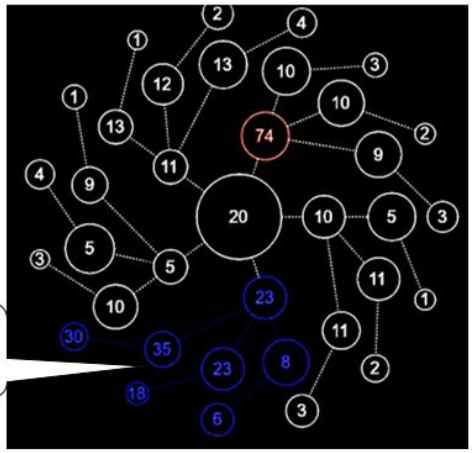
Aquí puede observarse como el Bot manda tropas hacia sus descendientes con el objetivo de conquistarlos.



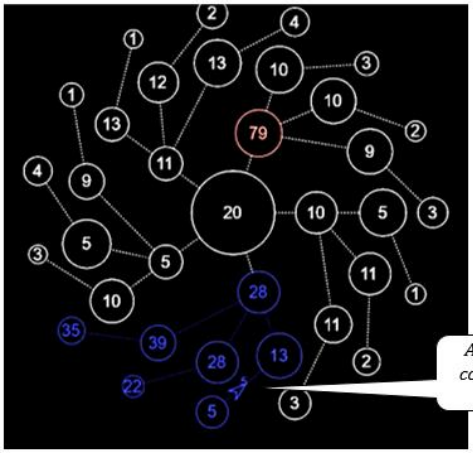
Ya conquisto dos de sus descendientes, ahora va por el resto.



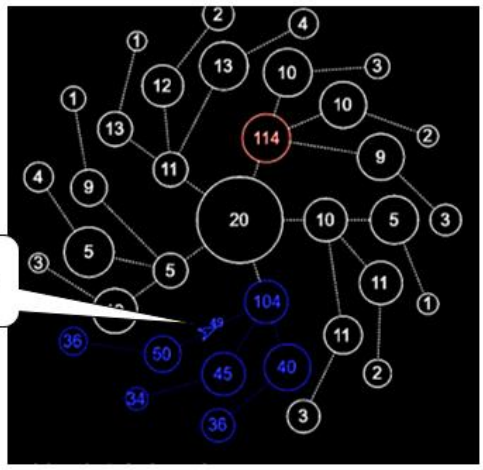
El Bot ya conquisto a todos sus descendientes. Ahora estos, deben reagrupar tropas en el Bot.



Ahora, los planetas conquistados envían tropas al Bot.



Conquistados reagrupan tropas en el Bot.



Resultados Obtenidos

El juego se probó en reiteradas oportunidades con el fin de constatar su correcto funcionamiento y si bien los resultados fueron satisfactorios, se notó una sutil deficiencia con respecto al retardo en la llamada al método CalcularMovimiento(), el cual es aproximadamente entre 2 y 4 segundos. Esto deja en claro, que sin importar cuantas veces se optimice el rendimiento de la estrategia, está siempre va a ser inferior a cualquier movimiento que pudiera realizar el jugador. No obstante, como se mencionó al principio de este apartado, los resultados fueron satisfactorios, ya que cumple con los requisitos solicitados. Es decir, el planeta del Bot conquista a sus descendientes y luego estos reagrupan sus tropas en el planeta conquistador.
Con respecto a las consultas, también el resultado fue el esperado, ya que cada vez que se inicia el juego, o se reposicionan los planetas del Bot y del Jugador, se actualizan las consultas mostrando la información pertinente de acuerdo a las nuevas características actuales del juego.

Conclusión

Para la realización del presente trabajo, se aplicaron varios conceptos vistos en la materia, se destaca por ejemplo la implementación de Arboles Generales y la manera de recorrer por niveles este tipo de estructura de datos. Estos conceptos fueron aplicados sin mayores complicaciones y si bien el desarrollo de la estrategia presento algunas dificultades, se lograron sortear estos obstáculos de manera satisfactoria. Es importante también mencionar que la catedra proporciono todas las herramientas y recursos necesarios para poder llevar a cabo la realización del presente trabajo.

El aprendizaje de las diferentes estructuras de datos, como así también los conceptos relacionados con complejidad temporal, son de una gran importancia en el ámbito del desarrollo de software. Gracias a este trabajo Practico se pudieron afianzar varios de esos conceptos, de manera que el proceso de desarrollo del presente, fue muy alentador.