Universidad Nacional Arturo Jauretche

Instituto de ingeniería y agronomía

Complejidad temporal, estructura de datos y algoritmos

Informe Trabajo Practico Final

Alumno: Ruhl Matias Agustin

Comisión 4

Dni: 43387629

Profesor: Alejandro Fontan

2do cuatrimestre 2020

Índice

Introducción

En el presente informe, se llevará a cabo el desarrollo del Trabajo Practico Final de manera teórica, demostrando y explicando los correspondientes métodos creados. Estos mismos estarán separados y diferencias, además, se agregará el diagrama UML para ver de una forma mas simplificada las clases y métodos mas utilizados a lo largo de la realización del trabajo.

En segundo lugar, se expondrán algunas capturas demostrando todo el proceso que se llevo a cabo. Principalmente, nos centraremos en los siguientes métodos:
“CalcularMovimiento(ArbolGeneral<Planeta>arbol),Consulta1(ArbolGeneral<Planeta>arbol), Consulta2(ArbolGeneral<Planeta> arbol), Consulta3(ArbolGeneral<Planeta> arbol) “ .

Por último, se presentarán los problemas encontrados, confusiones/dudas, que pudieron surgir al momento de crear dichos métodos, con sus respectivas posibles soluciones. Una vez presentado esto, se hará un comentario personal sobre todo el Trabajo y la experiencia obtenida.

Creando métodos

En esta instancia se agregarán unas capturas del desarrollo de los métodos solicitados por la catedra: (Consulta1, Consulta2, Consulta3 Y CalcularMovimiento).

Consulta1

En primer lugar, se referirá al método “Consulta1”:

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

        string ms = "" ;

        int distancia = Distancia( arbol , listaPlanetaa ) ;

        ms += "La distancia entre la raiz del arbol y del nodo mas cercano al BOT " + "es de : " + distancia ;

        return ms ;
}
```

Método auxiliar que calcula la distancia hacia un nodo que pertenezca a la Inteligencia Artificial.

```
public int Distancia ( ArbolGeneral<Planeta> arbol ,  List<Planeta> lista  ) {

        int distancia = 0 ;
        if ( arbol.getDatoRaiz().EsPlanetaDeLaIA() == true  ) {
                return distancia ;
        }
        else{
                lista.Add( arbol.getDatoRaiz() ) ;
                foreach ( ArbolGeneral<Planeta> planetaActual in arbol.getHijos() ) {
                        Distancia(planetaActual , lista) ;
                        return distancia++ ;
                }
```

En este primer caso, debemos retornar la distancia entre la raíz del árbol recibido por parámetro y el nodo más cercano al BOT. Para ello, lo que debemos hacer es, ir recorriendo el árbol desde la raíz hasta algún nodo que encontremos que pertenezca a la IA. Se me ocurrió usar un recorrido Pre-Orden ya que comienzan en la raíz.

En el método auxiliar llamado “Distancia” recibimos un árbol y una lista de planetas y lo que realizamos es recorrer el árbol de forma recursiva por medio del “Foreach”, siempre y cuando la raíz del árbol no pertenezca a la IA.

Para saber cual es el nodo más cercano al BOT, lo que hacemos es ir recorriendo los hijos del árbol, y si no pertenecen a la IA, aumentamos a la distancia.

Dicho procedimiento se ejecutará siempre y cuando el nodo actual NO sea de la inteligencia artificial.

Consulta2:

En segundo lugar, explicaremos brevemente la consulta2 :

```
public String Consulta2( ArbolGeneral<Planeta> arbol)
{
        Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>() ;
        cola.encolar(arbol) ;
        int lv = 0 ;
        string ms = "" ;
        while(!cola.esVacia()) {
                int elem = cola.cantElementos() ;
                lv++ ;
                int cantidad = 0 ;
                int poblacionPorLv = 0 ;
                while (elem-- > 0 ) {
                        ArbolGeneral<Planeta> nodoActual = cola.desencolar() ;
                        if ( nodoActual.getDatoRaiz().Poblacion() > 10 ) {
                                cantidad++ ;
                        }
                        foreach ( ArbolGeneral<Planeta> nodoHijo in nodoActual.getHijos() ) {
                                cola.encolar(nodoHijo) ;
                        }
                }
                ms += "Nivel " + lv + ": " + cantidad + " ." ;
        }
        return ms ;
```

Observamos que el método retorna la cantidad de planetas que tienen una población mayor a “10”. Dicha restricción la obtenemos haciendo “IF (nodoActual.getDatoRaiz().Poblacion() > 10)”.

Mientras que disminuimos a “elem” que contiene la cantidad de elementos que hay actualmente en la cola, siempre y cuando sea mayor a “0”. Si se cumple la restricción aumentamos la Cantidad, en cambio, sino se cumple, encolamos el “nodoHijo” del nodo que actualmente estamos recorriendo. El método se ejecutará siempre y cuando la cola no sea vacía.

Consulta3:

En tercer lugar, nos enfocaremos en la Consulta3:

```
public String Consulta3( ArbolGeneral<Planeta> arbol)
{
        Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>() ;
        cola.encolar(arbol) ;
        int lv = 0 ;
        string ms = "" ;
        while(!cola.esVacia()) {
                int elem = cola.cantElementos() ;
                lv++ ;
                int cantNivel = 0 ;
                int poblacionPorLv = 0 ;
                while (elem-- > 0 ) {
                        ArbolGeneral<Planeta> nodoActual = cola.desencolar() ;
                        cantNivel++ ;
                        poblacionPorLv += nodoActual.getDatoRaiz().Poblacion();

                        foreach ( ArbolGeneral<Planeta> nodoHijo in nodoActual.getHijos() ) {
                                cola.encolar(nodoHijo) ;
                        }
                }
                ms += "Nivel " + lv + ": " + poblacionPorLv/cantNivel + ".";
        }
        return ms ;
```

En este caso, debemos retornar el promedio poblacional por nivel. Dicho promedio lo obtenemos haciendo “poblacionPorLv / cantNivel”. Para hallar ambas expresiones, realizamos el siguiente procedimiento:

En primer lugar, la “poblacionPorLv” la calculamos cuando obtenemos la Población del nodo que estamos recorriendo actualmente. Esto lo vemos en el código como: “poblacionPorLv += nodoActual.getDatoRaiz().Poblacion() ; , donde iremos sumando la población por nivel del árbol recibido por parámetro.

En segundo lugar, a “cantNivel” la obtenemos cada vez que sumamos “1” al contador, esto ocurre cada vez que recorremos un nivel, donde posteriormente obtenemos su población.

Calcular Movimiento

Por último, debemos situarnos en la explicación del método: “CalcularMovimiento”.

Para comenzar, he tenido muchos problemas para desarrollar este método, pude crear varios pero ninguno me daba resultado en la ejecución del juego (el Bot a comenzar la partida no avanzaba hacia la raíz).

A continuación, dejare unas capturas de dichos métodos.

Forma 1:

```
public Movimiento CalcularMovimiento (ArbolGeneral<Planeta> arbol)
{
        //debemos encontrar el nodo q pertenece al BOT

        List<Planeta> listaPlanetas = new List<Planeta>() ;

        if ( arbol.getDatoRaiz().EsPlanetaDeLaIA() == false ) {

                List<Planeta> caminoHaciaIA = BusquedaIA(arbol,listaPlanetas) ;

                Movimiento movHaciaIA ;

                for ( int i = 0 ; i < caminoHaciaIA.Count ; i++ ){

                        movHaciaIA.origen = caminoHaciaIA[i-1] ;
                        movHaciaIA.destino = caminoHaciaIA[i] ;

                }
                return movHaciaIA ;
        }
        else {

                List<Planeta> caminoHaciaJugador = AtaqueJugador(arbol,listaPlanetas) ;
                Movimiento movHaciaJugador ;

                for ( int i = 0 ; i < caminoHaciaJugador.Count ; i++ ){

                        movHaciaJugador.origen = arbol.getDatoRaiz() ;
                        movHaciaJugador.destino = caminoHaciaJugador[i] ;

                }
                return movHaciaJugador ;
        }
```

Métodos auxiliares:

```
//Con este metodo armamos un camino hacia algun nodo de la IA
public List<Planeta> BusquedaIA ( ArbolGeneral<Planeta> arbol , List<Planeta> lista ) {

        if ( arbol.getDatoRaiz().EsPlanetaDeLaIA() == true  ) {
                return lista ;
        }
        else{
                lista.Add( arbol.getDatoRaiz() ) ;
                foreach ( ArbolGeneral<Planeta> planetaActual in arbol.getHijos() ) {
                        BusquedaIA(planetaActual , lista) ;
                        return null ;
                }
                //saco el ultimo elemento si no es de la IA

                if( lista[lista.Count - 1].EsPlanetaDeLaIA() == false ) {
                        lista.RemoveAt(lista.Count - 1) ;
                        return null ;
                }

        }
}
//Recorrido desde la raiz al Jugador
public List<Planeta> AtaqueJugador ( ArbolGeneral<Planeta> arbol , List<Planeta> lista ) {

        if ( arbol.getDatoRaiz().EsPlanetaDelJugador() == true  ) {
                return lista ;
        }
        else{
                lista.Add( arbol.getDatoRaiz() ) ;
                foreach ( ArbolGeneral<Planeta> planetaActual in arbol.getHijos() ) {
                        BusquedaIA(planetaActual , lista) ;
                        return null ;
                }
                //saco el ultimo elemento si no es de la IA

                if( lista[lista.Count - 1].EsPlanetaDelJugador() == false ) {
                        lista.RemoveAt(lista.Count - 1) ;
                        return null ;
                }
```

Esta fue una de las formas que se me ocurrió de desarrollar el método. Plantee dos métodos auxiliares llamados “BusquedaIA” y “AtaqueJugador”, para hacer estos métodos, me base en un recorrido Pre-Orden que comienza desde la raíz del árbol recibido por parámetro.

Ambos métodos retornan una lista de planetas, esta lista, es el camino que se llevo a cabo para llegar a un nodo perteneciente a la IA o de un Jugador. Una vez que el método termina, en el método “CalcularMovimiento” lo que hacemos es: si la raiz del árbol no pertenece a la IA, instanciamos como “caminoHaciaIA” a lo que retornamos del método “BusquedaIA”.

Una vez creado el camino lo que hacemos es crear el Movimiento llamado “movHaciaIA”, para luego establecer sus propiedades. Para ello, lo que hacemos, es por medio de un “For”, iterar sobre la lista: “caminoHaciaIA”, cuando estamos recorriendo la lista sabemos que el ultimo nodo (en este caso “i”) este seria el destino, y el origen el anterior al nodo encontrado.

Pero también, tenemos que pensar que cuando recorremos por completo una rama del árbol y NO encontramos algún planeta que pertenezca a la IA, lo que hacemos en este caso, es eliminar al último planeta recorrido, ya que, el padre de este nodo puede tener más hijos, y estos mismos nodos hijos pueden ser de la IA.

De esta misma forma, funciona el otro método llamado “AtaqueJugador”, salvando las diferencias, ya que, en este caso comparamos si los planetas pertenecen o no a un Jugador.

Forma 2:

```
public Movimiento CalcularMovimiento (ArbolGeneral<Planeta> arbol)
{
if ( arbol.getDatoRaiz().EsPlanetaDeLaIA() == false ) {

        //armamos un camino hacia un nodo de la IA

        List<Planeta> ListaPLanetas = new List<Planeta>() ;
        ListaPLanetas.Add(arbol.getDatoRaiz());
        Movimiento movAI  ;
        Movimiento movJugador  ;
        // comenzamos en la raiz del arbol
        movAI.origen = arbol.getDatoRaiz() ;
        movJugador.origen = arbol.getDatoRaiz() ;
        for ( int i = 0 ; i < ListaPLanetas.Count ; i++ ) {
                if ( ListaPLanetas[i].EsPlanetaDeLaIA() == true) {
                        movAI.destino = ListaPLanetas[i] ;
                        return movAI ;
                }
        }
        for ( int p = 0 ; p < ListaPLanetas.Count ; p++ ){
                if (  ListaPLanetas[p].EsPlanetaDelJugador()  == true ) {
                        movJugador.destino = ListaPLanetas[p] ;
                        return movJugador ;
                }
        }
        return null ;
}else {
                // si la raiz es planeta de la IA

                Movimiento movHaciaAI  ;
                movHaciaAI.origen = arbol.getDatoRaiz() ;
                movHaciaAI.destino = arbol.getDatoRaiz() ;

                return movHaciaAI ;
```

En este caso, observamos que no tenemos métodos auxiliares. Si no que tenemos en el mismo método “CalcularMovimiento” una restricción con un IF, donde si la raíz no pertenece a la IA, armamos un camino hacia algún planeta de la Inteligencia Artificial.

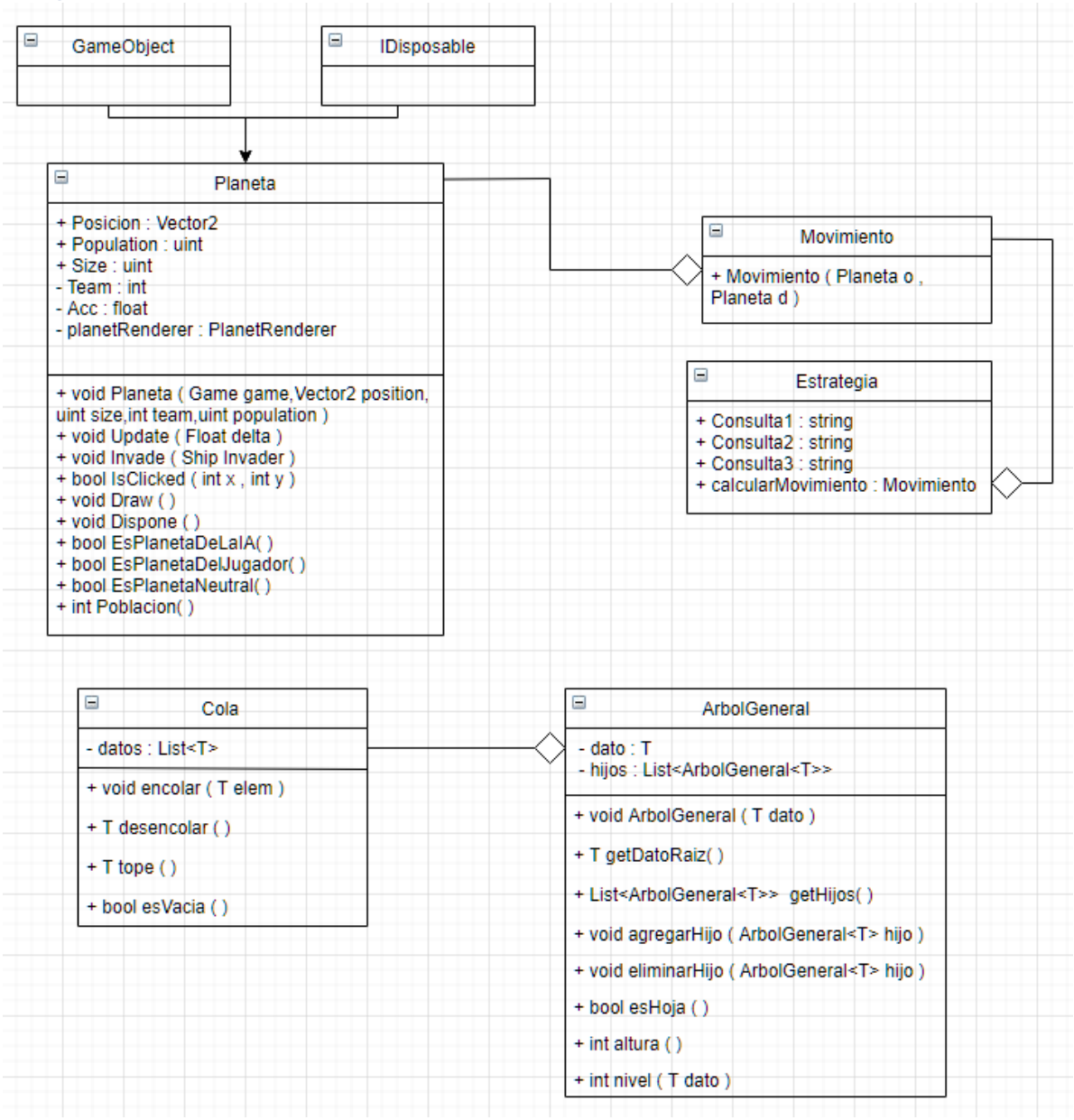
Posteriormente, observamos que creamos una lista de planetas, en la cual agregamos como primer elemento a la raíz del árbol. Luego, lo que hacemos, es crear dos instancias de la clase “Movimiento” una para la IA “movIA” y otra para el jugador “movJugador” y establecemos el origen para ambos.

Lo que nos falta es hallar el destino de los movimientos, para ello, realizamos un “For” en el cual iteramos sobre la lista de planetas creada “ListaPlaneta” y si encontramos que un planeta que pertenece a la IA o al Jugador ya podemos establecer el destino.

Por otro lado, también tenemos que contemplar el caso en el que la raíz sea de la IA, aunque sea muy poco probable, esto lo vemos en el código con el “Else”, donde creamos un nuevo movimiento llamado “movHaciaAI” el cual tiene como origen y destino a la raíz del árbol a recorrer.

Luego de exponer ambas formas de realizar el método “CalcualarMovimiento”, decidí dejar la primera forma, porque, me pareció mucho más cómodo de realizarlo de esa manera. Estos métodos auxiliares me ayudaron mucho a la organización del código, pero no se si es la forma correcta de resolver el ejercicio, seguramente no lo sea, pero definitivamente me convenció mucho más la primera.

Diagrama UML



Repositorio

https://github.com/Matiasruhl/TP_Final

Conclusiones

Luego de haber concluido con lo solicitado en el Trabajo Practico Final de la materia, he notado que gracias a como esta organizado y en la manera que hay que realizar los métodos, que obtuve un poco mas de experiencia en cuanto a los recorridos de árboles. Sinceramente, antes se me complicaba con el recorrido pre-orden pero en este trabajo pude despejar estas dudas que me quedaron de la cursada.

Una vez dicho esto, a medida que fui desarrollando los métodos, me encontré con infinidad de problemas, con los "*using*" de algunas clases, no podía cargar los *commits* en Git Hub. Estos problemas que no fueron de gran importancia, pude solucionarlos fácilmente con un cambio de proyecto y por otro lado, volviendo a repasar unos conceptos de como operar en GitHub desde el escritorio.

En cambio, hay que destacar que como bien puse en el método "CalcularMovimento" se me ocurrieron dos formas de como hacer el ejercicio, terminé decidiéndome por la primera forma, ya que, fue la más organizada, y donde pude obtener los caminos para llegar al "BOT" o la "Usuario". Se ve que me faltaron pequeños detalles para que funcione correctamente. Me gustaría ver como se resuelve correctamente el método "CalcularMovimiento", para poder así, encontrar mi error, y optimizar mucho mas mi TP.

Por otro lado, en cuanto a la Consulta1, ya la había desarrollado hace varios días, pero al concluir con el ultimo método("CalcularMovimiento") me sentía capas de poder optimizarlo aún más, esto se podrá ver en los "*commits"* del repositorio que adjuntare a este archivo.

Vale aclarar, que el repositorio, están TODOS los cambios realizados en el código, en los más importantes (Donde se cambiaron muchas líneas de código) se añadió una breve descripción sobre lo agrego y/o eliminado de los métodos en cuales se estaba trabajando.

Como propuesta a futuro, estaría bueno que se tenga en consideración la utilización de mas clases para desarrollar los métodos, porque de esta forma, podemos establecer las áreas de trabajo sin mezclar atributos o alguna clase.

Para concluir y no menos importante, siento que las consignas de los métodos a desarrollar no son del todo claras, en varios casos se nos dice que lo que recibimos es una raíz, pero en otros es un árbol completo, esta parte fue media confusa, pero se podría evitar este inconveniente.