

Trabajo Final

Renzo Martinez

Complejidad Temporal, Estructuras de Datos y Algoritmos

Comison 3

10/11/2021

# Diagrama UML

Como verán en el archivo adjunto en este mismo repositorio, esta muestra el diagrama UML con las clases asociadas a la estrategia ya que este mismo depende de cada una para buscar la información y poder hacer una respuesta con ellos.

# Detalles de la implementacion

Para esta estrategia me aventure en diferentes metodos, siendo algunos

* de busqueda
* de manipulacion de listas
* calculo de movimiento
* etc.

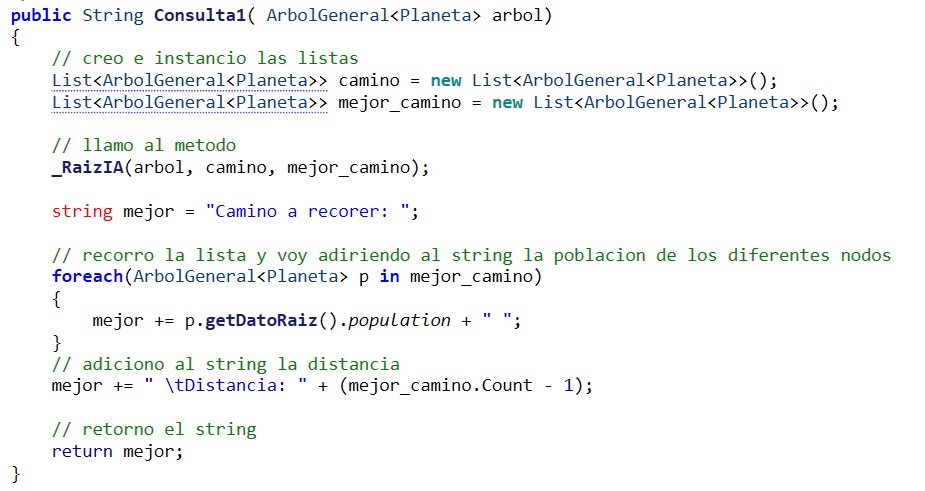
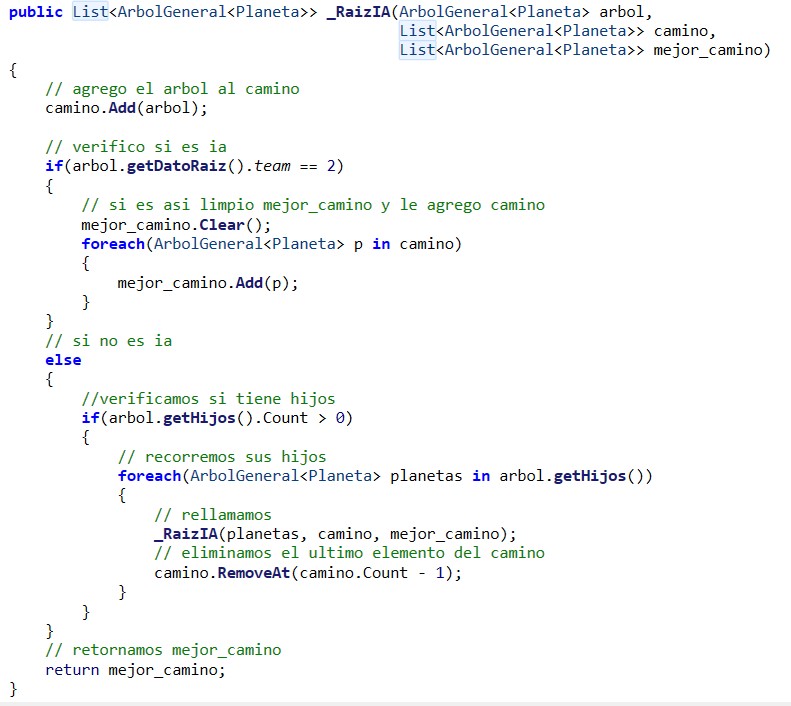
A medida de que lo implementaba me encontre con varios retos y errores.

* Los errores con los que me tope en Consulta1() fue que quise implementar busqueda por niveles pero me parecia algo poco accesible, asi que pense en usar algo similar al DFS usado en los grafos que usa varias listas para su implemetacion siemdo su unico cambio el no usar una lista de booleanos para marcar su camino.
* Los errores con los que me tope en Consulta2() no me encontre con mucha dificultad despues de haber terminado el metodo anterior ya que pense que este serialo inverso, donde en vez de almacenar hasta encontrar el planeta de la ia, una vez encontrado dicho planeta recorrer sus hijos y almacenarlos.
* Los errores con los que me tope en Consulta3()

Como mi idea en este era obtener la lista de por niveles y despues recorrerla de acuerdo a ello.

Para hacer la lista no tuve problemas ya que use la busqueda por niveles pero tuve algunos percances a la hora de manipular la lista pero lo solucione agregando multiples variables de apoyo.

# Codificado y Descripcion de sus Partes



Como veremos en las imágenes, requerí de ayuda de otro método llamado “\_RaizIA” para poder solucionar la consulta pedida.

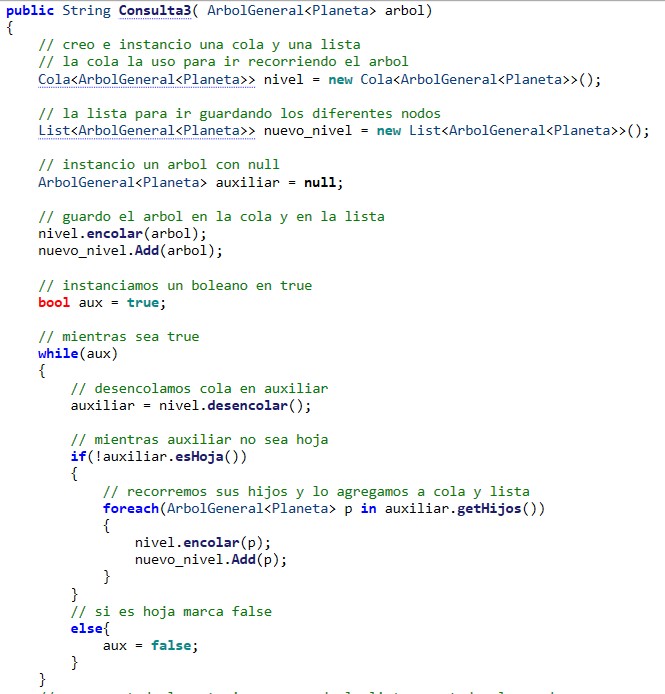
Este método lo que hace es recorrer el árbol e ir agregando sus hijos a una lista llamado “camino”, si se encuentra al hijo que sea de la ia se copiara los elementos de camino a otra lista llamada “mejor\_camino” y sino se elimina el ultimo elemento en camino. Una vez recorrido todo se retornará mejor\_camino.

Después en el método Consulta1 simplemente recorreremos la lista e iremos agregando la información al texto que luego será retornado.

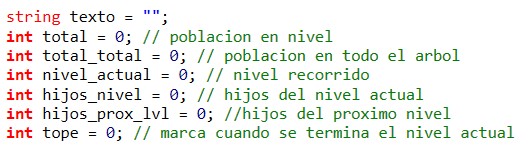
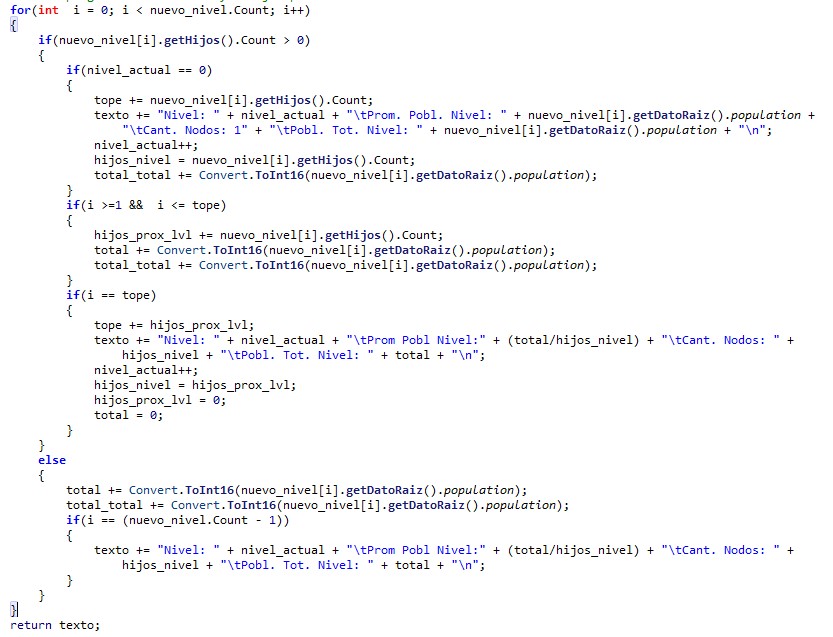


Para la Consulta2 me ayude también con otro método que también retorna una lista y luego uso esta para recorrerla y agregar sus datos al string

El método “\_Consulta2”, para este me utilice la lógica inversa de “\_RaizIA” y lo adapte para que al encontrar al hijo que pertenezca a la ia guarde a sus hijos en la lista y la retorne.



Para esta solución me inspire en la búsqueda por niveles donde recorre el árbol y con ayuda de una cola y un árbol auxiliar va imprimiendo los datos consultados, pero en vez de imprimirlos los guardaba en una lista aparte que la llame “nuevo\_nivel”.

Con la anterior lista me dispuse a recorrerlo de a rangos y como tenia que retornar un string me puse en uso de varias variables para ir marcando diferentes cosas las cuales están comentados en cada línea.

Lo que hice en la imagen anterior fue que en la primera iteración ya que este iba a ser el nodo principal simplemente guarde la información en texto y cambie el tope sumándole la cantidad de hijos también guarde sus hijos en “hijos\_nivel”.

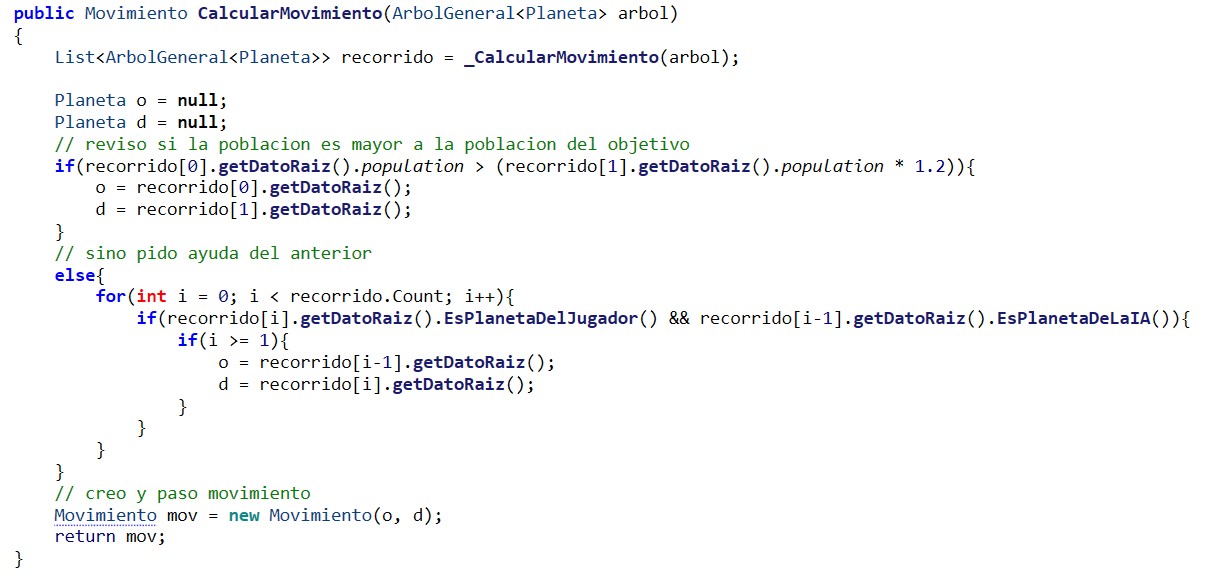
En las siguientes iteraciones sumaria los hijos de los nodos del nivel en “hijos\_prox\_lvl” como también su población en “total”.

Cuando la iteración “i” llegue a “tope”, se guardaría la información recopilada del nivel en “texto” reemplazamos “hijos\_nivel” por “hijos\_prox\_lvl” y a este ultimo junto con total lo seteamos en cero.

A todo lo anterior se lo ejecutaba mientras el árbol de planetas que se esté leyendo tenga hijos (o no fuese hoja).

Cuando se llegaban a las hojas (o el ultimo nivel del árbol) simplemente hacemos la cuenta de la población de cada hoja y al llegar a la última posición de la lista guardar la información en “texto”.

Después del ciclo for retornaremos el “texto”.



Para la solución al método Consulta3 realice dos métodos apartes que ya los iré comentando lo cual para ahora sirve decir que retorna la lista desde el nodo que tiene a la ia hasta el nodo que tenga el jugador.

Se instancia planeta origen (o) y destino (d) en null.

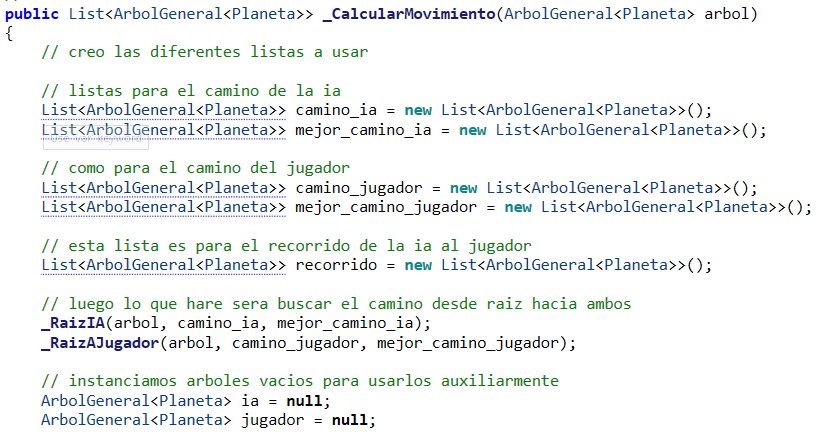
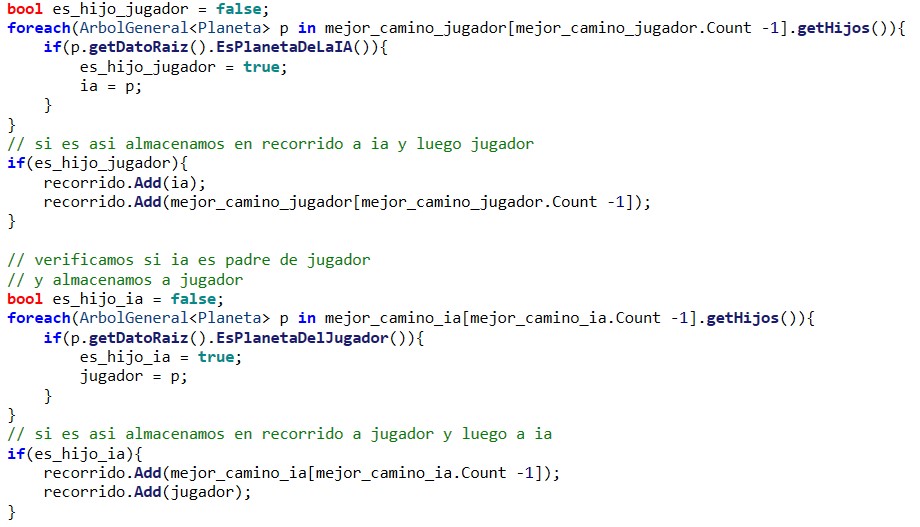
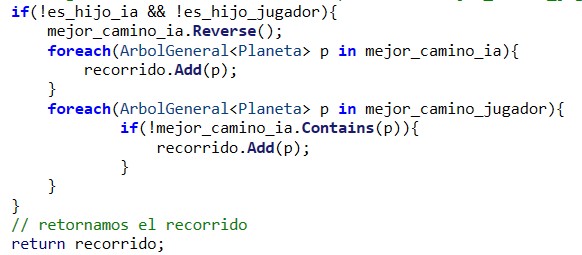
Con dicha lista (la cual el primer índice es siempre el planeta de la ia) vemos si la cantidad de población del planeta ia es mayor a la cantidad de población del siguiente planeta.

Si es así se agrega la ia en origen y el otro planeta como destino.

Sino esperara a juntar población.

En el caso de que se encuentre con un planeta del jugador se pedirá ayuda al anterior planeta que la ia haya conquistado, para ello se recorrerá la lista y si la iteración es planeta del jugador y su anterior en la lista es planeta de la ia entonces se pondrá origen como el anterior y destino el planeta de la ia.

Luego se crea una nueva instancia de movimiento con o y d y se lo retorna.



Lo que hacemos aquí es manipular las listas que nos retornan los métodos “\_RaizIA” cuya lista es el recorrido de la raíz hasta la primera aparición de la ia y “\_RaizJugador” hace lo mismo, pero con el jugador.

Lo que hago después es revisar si uno es hijo del otro .

Para ello recorro la lista de hijos del planeta ia y reviso si pertenecen al jugador marco verdadero y almaceno en jugador a dicho nodo.

Si es jugador es hijo de ia agrego el último elemento de la lista y lo agrego a “recorrido”, luego agrego el jugador.

Si resulta ser hijo del jugador hago lo mismo pero esta vez buscando a ia en el hijo de jugador.

Si resulta no ser el hijo de ia ni de jugador simplemente doy vuelta la lista de la ia y la agrego a recorrido y luego recorro la lista de jugador y si no se repiten elementos lo agrego a recorrido.

Al finalizar todo este proceso devuelvo la lista recorrido

La anterior imagen muestra el código que recorre el árbol y ca almacenando la información obtenida, una vez que encuentre al jugador copiara la lista en otra y la retornara. De esta forma obtenemos el recorrido desde la raíz del árbol hasta el nodo jugador.

# Sugerencias de Mejora

Con respecto a las mejoras lo mas probable es que haya, pero no dispongo del conocimiento como para notar y hacer algo mejor.

Mediante las diversas pruebas que iba haciendo mientras ideaba una forma de realizar el proyecto me di cuenta que la formula del movimiento podría mejorarse, pero no se me ocurre algo mejor a lo que había implementado.

# Conclusión

En conclusión, el proyecto se pudo llevar a cabo con altibajos, pero pudo completarse con rango de margen de tiempo para una consulta. Al principio se idearon varias cosas, pero al ver como funcionaban y su cantidad de código fueron quitadas, por este motivo solo puedo esperar una devolución para ver si se tienen que eliminar o cambiar las cosas