

Trabajo Practico Final

Conquista Planetaria

Renzo Martinez

Complejidad Temporal, Estructuras de Datos y Algoritmos

Comison 3

29/11/2021

Contenido

[1. Introducción 2](#_Toc89102785)

[2. Diagrama UML 2](#_Toc89102786)

[3. Detalles de la implementacion 3](#_Toc89102787)

[4. Capturas del Juego 3](#_Toc89102788)

[5. Codificado y Descripcion de sus Partes 4](#_Toc89102789)

[6. Sugerencias de Mejora 12](#_Toc89102790)

[7. Conclusión 12](#_Toc89102791)

# Introducción

Una empresa informática está llevando a cabo un juego de computador que se desarrolla como una conquista planetaria donde el enfrentamiento se da únicamente entre dos jugadores

En dicho juego los planetas albergan una cantidad específica de naves que pueden pertenecer al jugador, al oponente o ser neutrales

La propiedad está representada por un color asignado:

* Rojo para el usuario
* Azul para el Bot
* Blanco para indicar neutralidad

Cada planeta tiene un conjunto de rutas interplanetarias que los interconecta y que sirven para enviar flotas de un planeta hacia otro.

En cada turno, el número de naves alojadas en los planetas del jugador y el Bot se incrementa de acuerdo a la tasa decrecimiento de cada planeta.

Los planetas neutrales no agregan nuevas naves a la flota que alojan hasta que sea conquistado por algún jugador.

Los movimientos serán posibles siempre y cuando exista una ruta interplanetaria

Cuando la flota llega un planeta enemigo o neutral, tiene lugar un enfrentamiento, en el cual resulta victoriosa aquella flota que albergue más naves

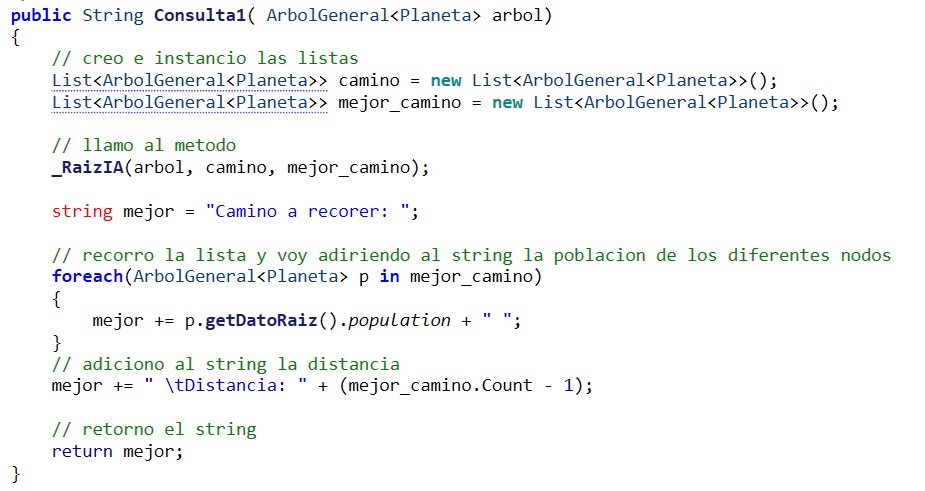
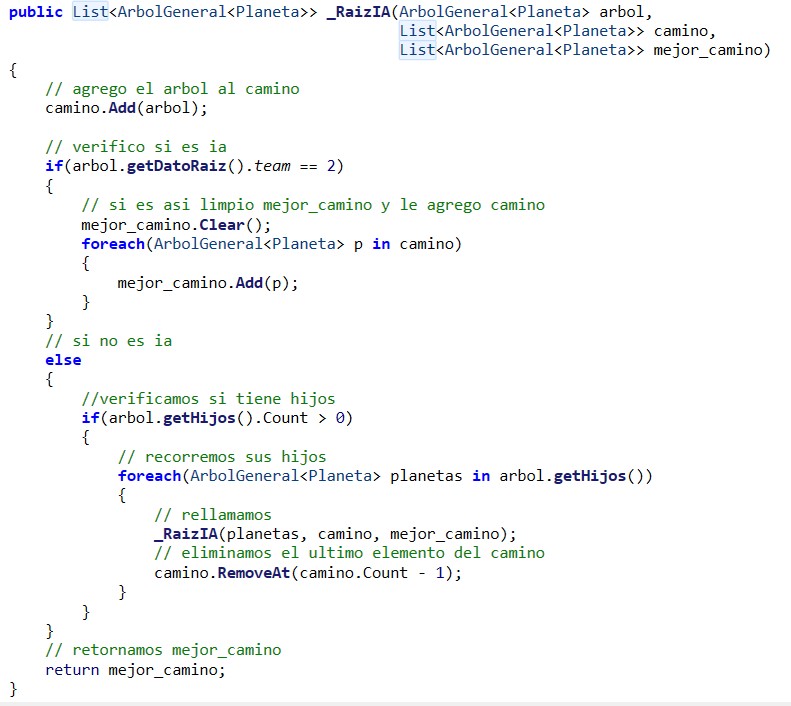
En caso de que el planeta de destino sea del propio jugador, ambas flotas se unen

La empresa nos contratado para implementar la estrategia de ataque que utiliza el Bot cuando en el mapa se disponen los planetas jerárquicamente o en forma de árbol general.

Esta estrategia tiene como objetivo atacar a un planeta del enemigo cuando es el turno del Bot.

En el juego la codificación de la estrategia antes descripta se realizará en la clase Estrategia.

# 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.

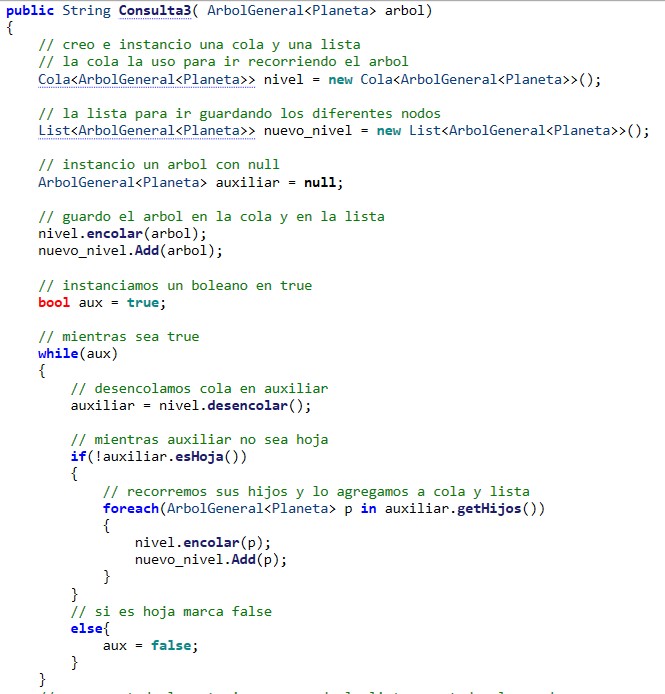
Este método lo que hace es recorrer el árbol e ir agregando sus hijos a una lista llamado “camino”, si se encuentra al hijo que sea de la ia se copiara los elementos de camino a otra lista llamada “mejor\_camino” y sino se elimina el ultimo elemento en camino. Una vez recorrido todo se retornará mejor\_camino.

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

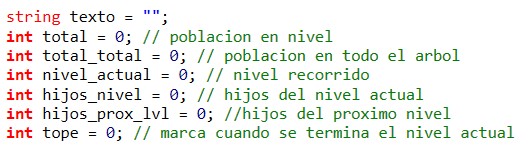
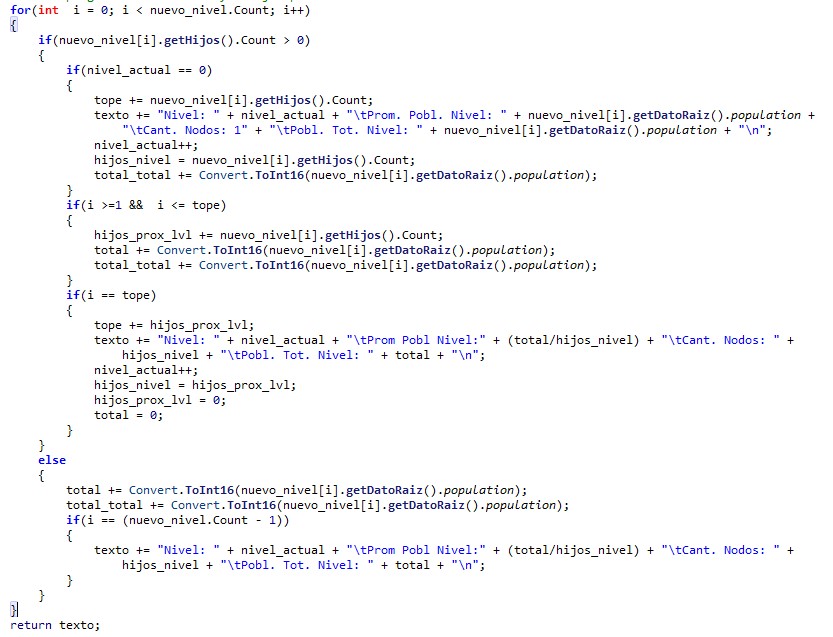


Para la Consulta2 me ayude también con otro método que también retorna una lista y luego uso esta para recorrerla y agregar sus datos al string

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(“Consulta3”) 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”.

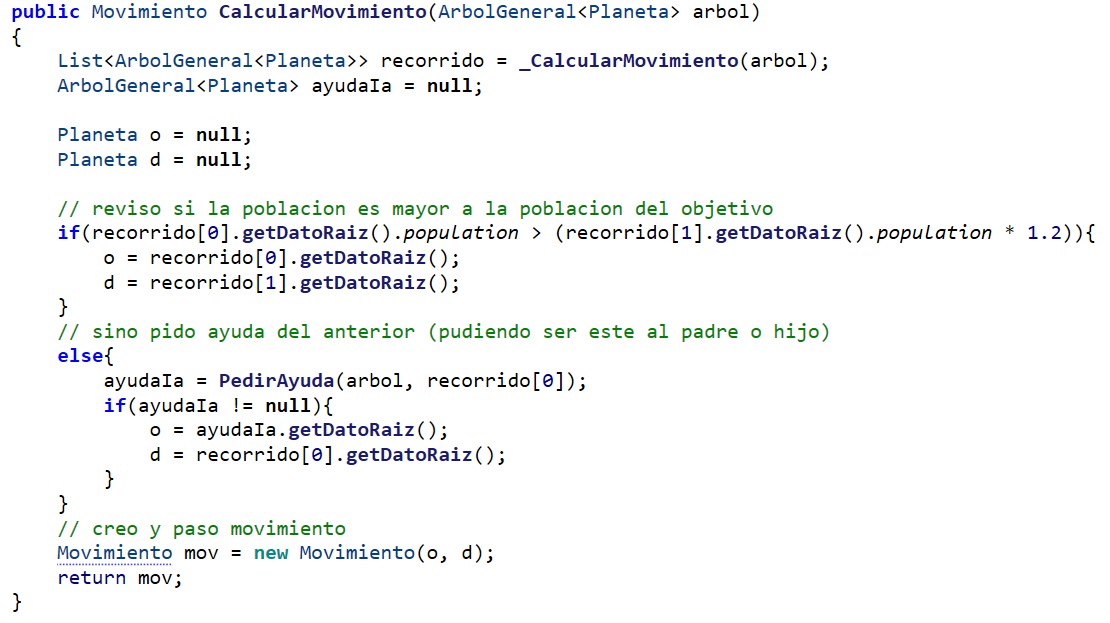
En las siguientes iteraciones sumaria los hijos de los nodos del nivel en “hijos\_prox\_lvl” como también su población en “total”.

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 último 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 CalcularMovimiento 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.

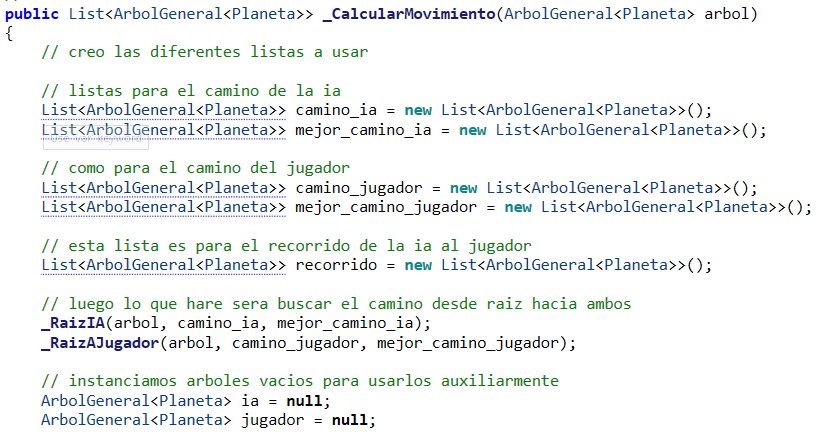
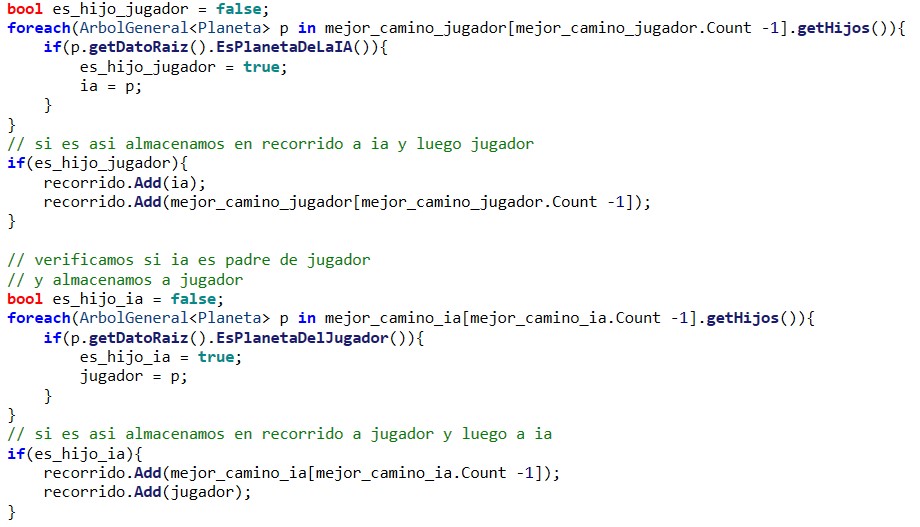
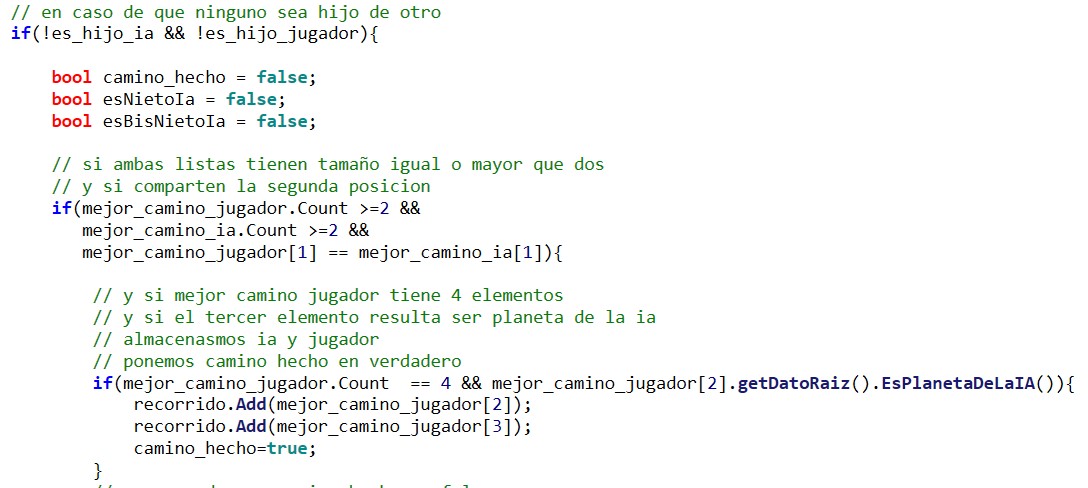
Se instancia planeta origen (o) y destino (d) en null.

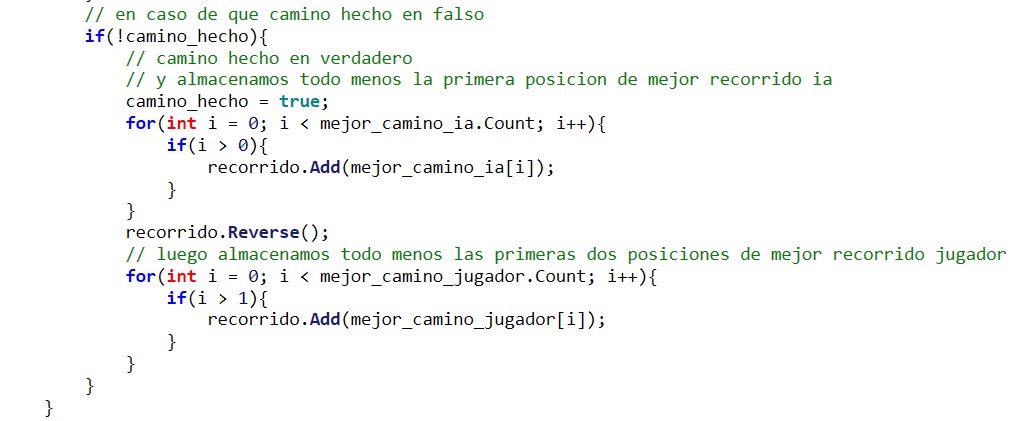
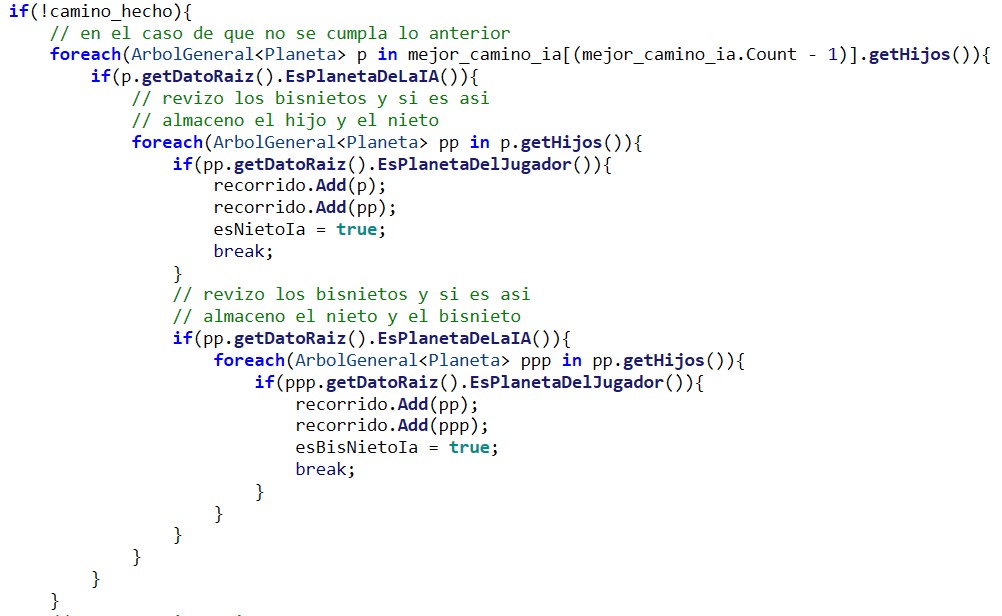
Con dicha lista (la cual el primer índice es siempre el planeta de la ia) vemos si la cantidad de población del planeta ia es mayor a la cantidad de población del siguiente planeta.

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

Sino se llamará al método PedirAyuda que busca al planeta padre o hijo que sean también de ia. Poniendo como origen dicho planeta y destino como 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 inicializo en false algunas variables que iré usando

Verifico si “mejor\_camino\_ia” y “mejor\_camino\_jugador” tienen dos o más elementos en la lista y verifico si el segundo elemento de ambas es lo mismo

* Si es así verifico si “mejor\_camino\_jugador” tiene cuatro elementos y si el tercero pertenece a la ia, siendo así simplemente agregaremos en recorrido sus últimos dos elementos y seteamos “camino\_hecho” en true

Si después de ello “camino\_hecho” sigue en false, verificamos si el jugador es nieto o bisnieto de ia para ello hacemos las respectivas búsquedas y una vez encontremos agregamos los dos elementos correspondientes (siendo ia-hijo o hijo-nieto) y detenemos la búsqueda.

Después se retornará la lista “recorrido” en el caso de que sea nieto o bisnieto.

En el caso de que no sea ninguno doy vuelta la lista “mejor\_camino\_ia” y agrego sus elementos a “recorrido” luego reviso los elementos de la lista “mejor\_camino\_jugador” y agrego en “recorrido” los elementos no recorridos.

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.

# Diagrama UML

Como verán, en el diagrama se hace muestra de como se relacionan las clases pre armadas con anterioridad con la estrategia desarrollada.

# 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.

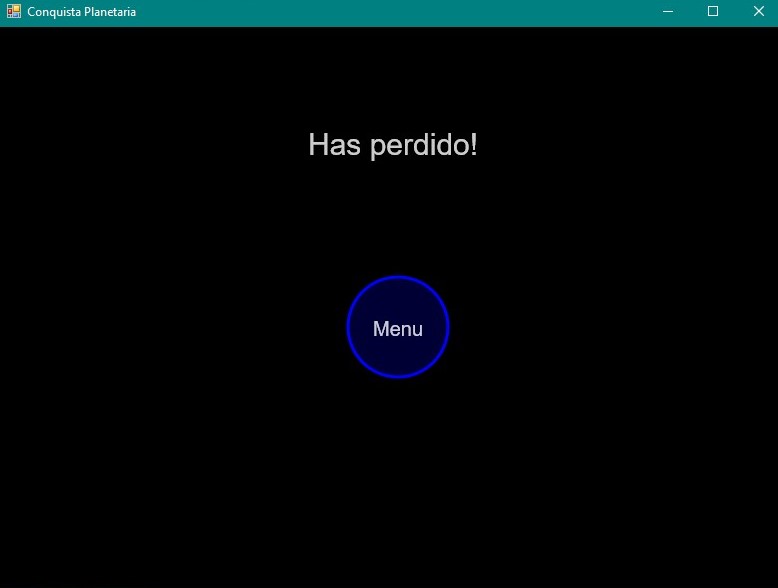
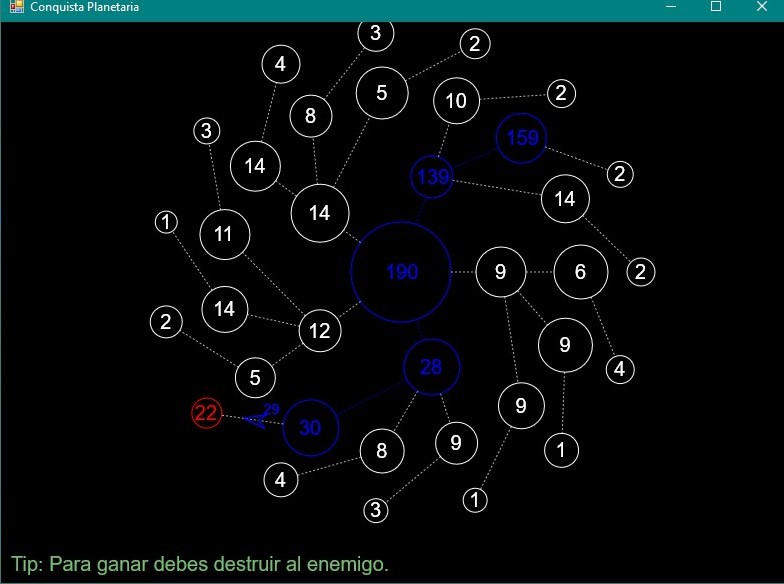
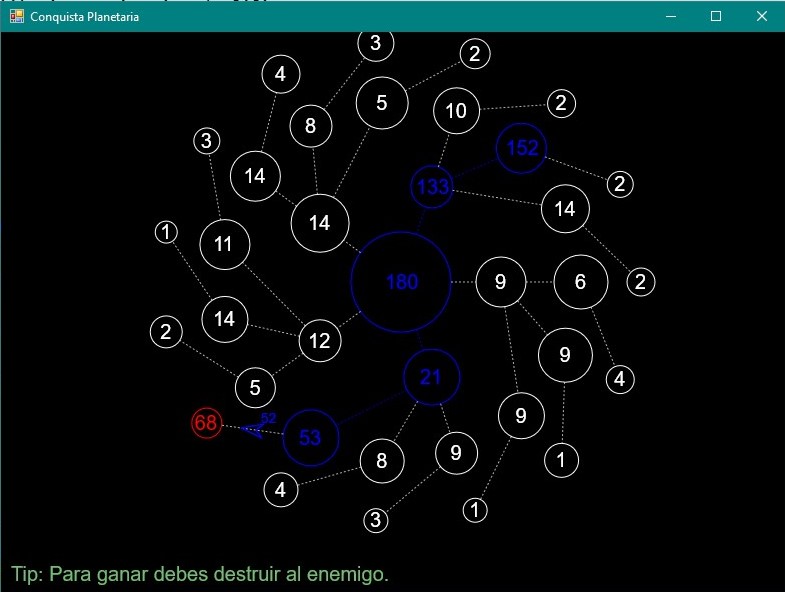
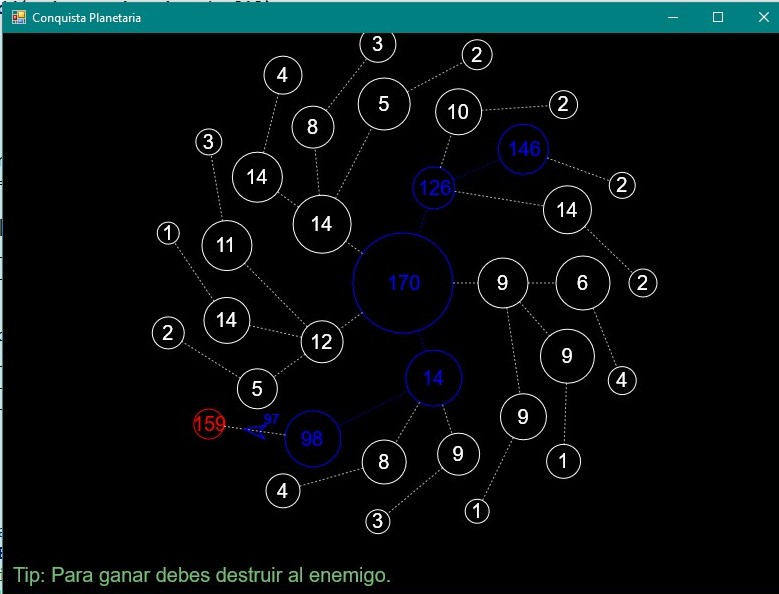
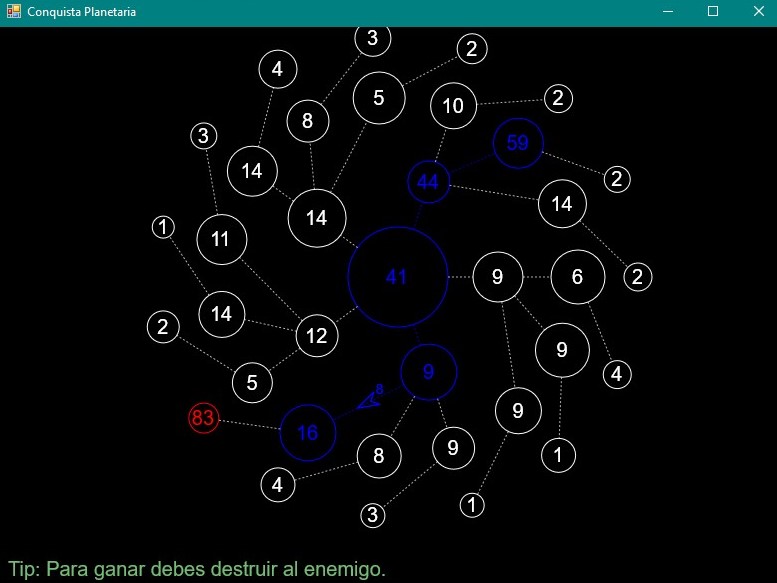
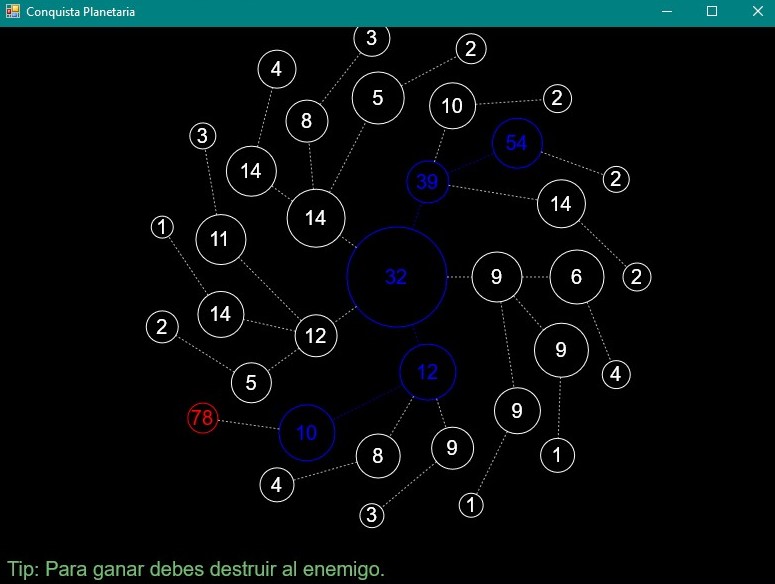
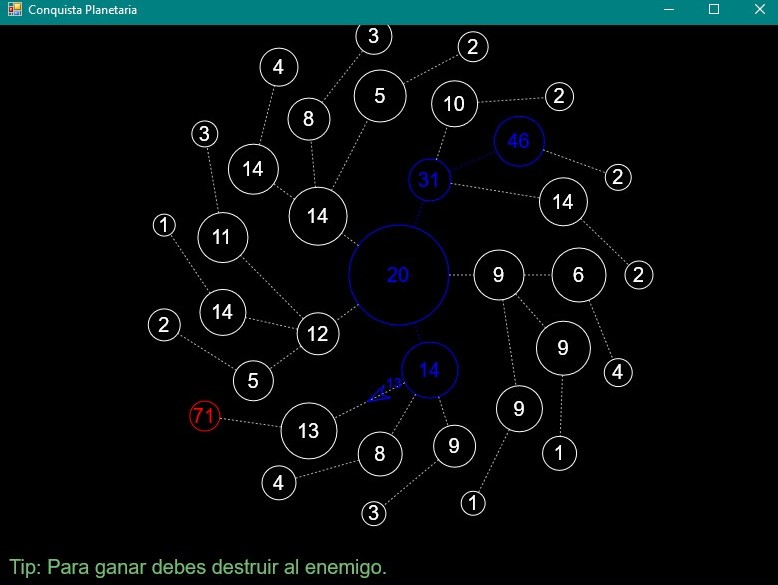
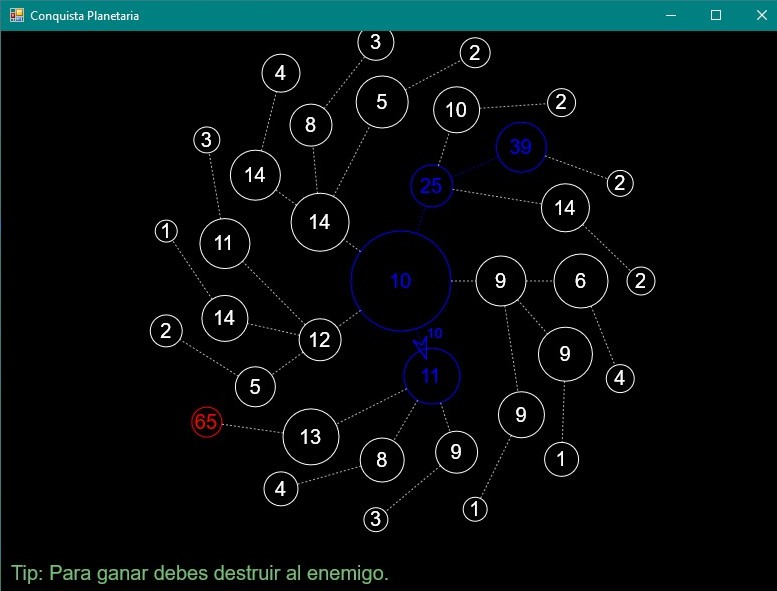
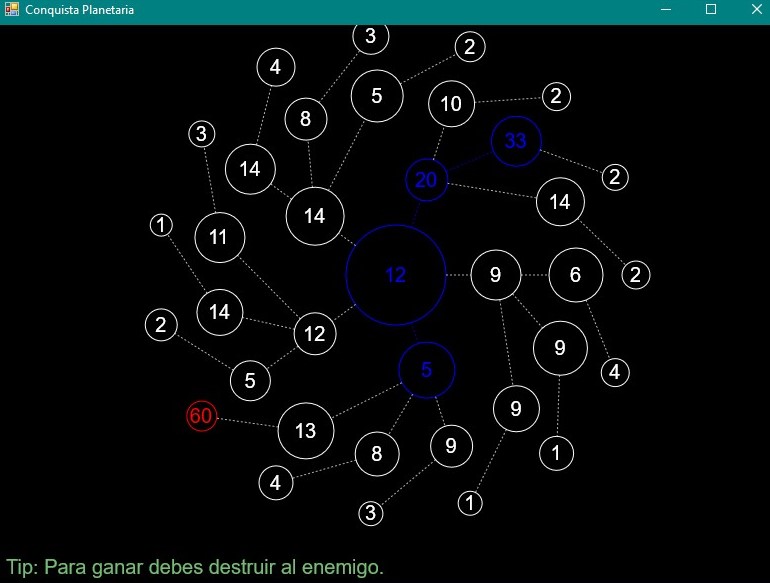
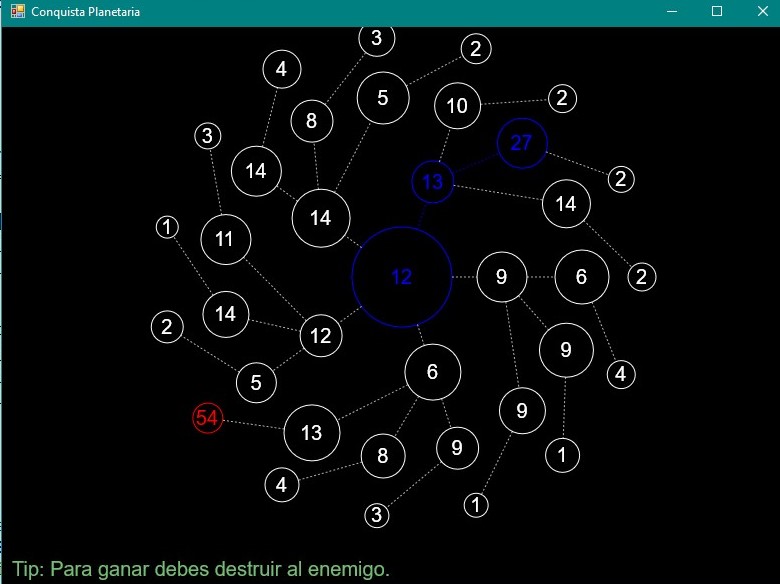
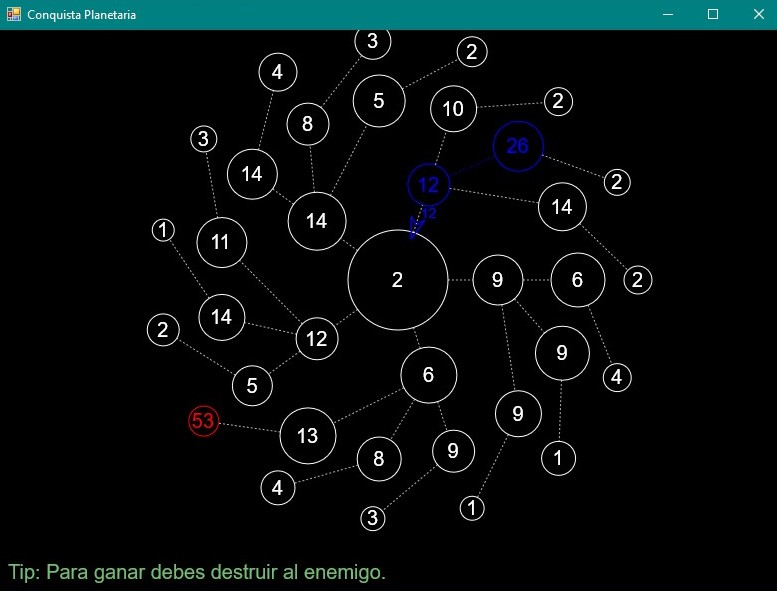
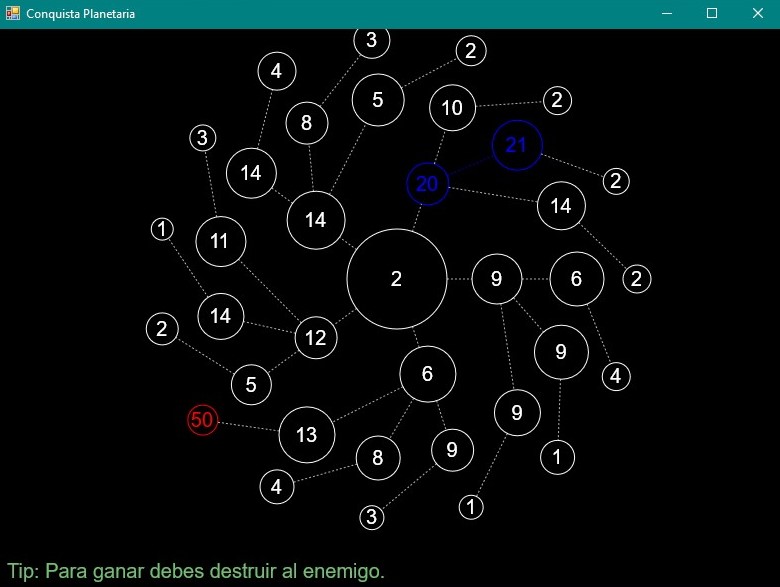
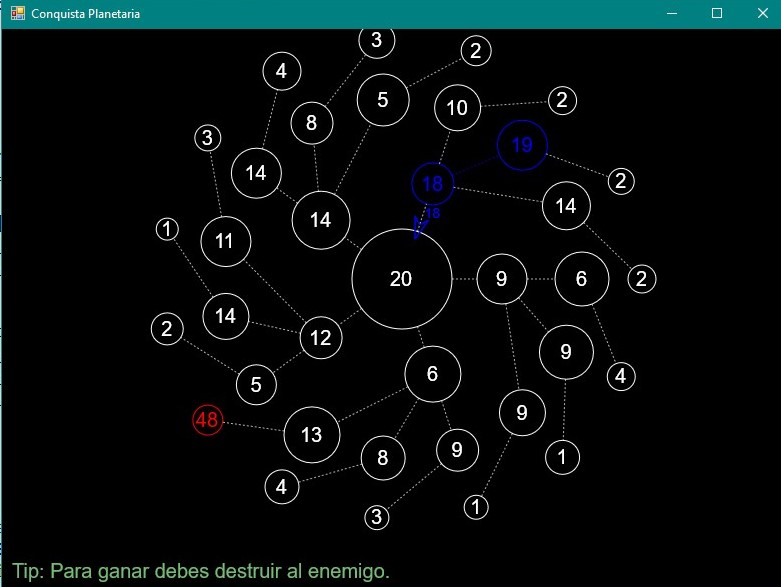
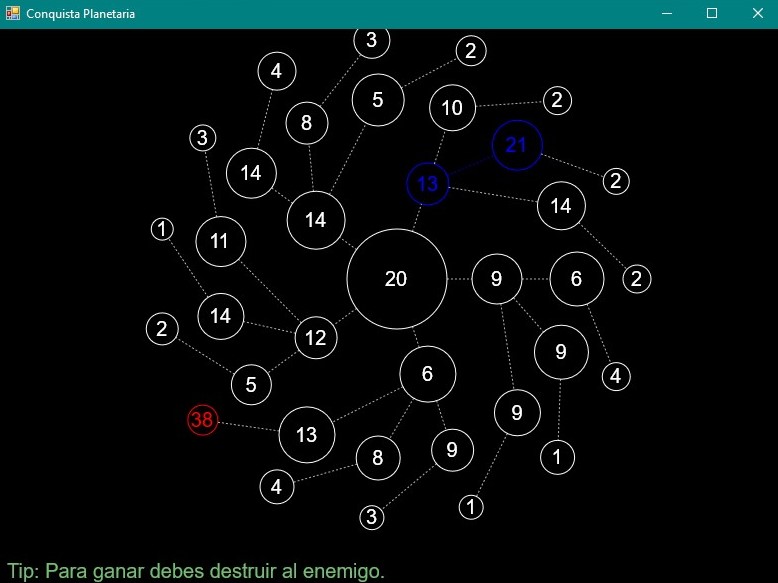
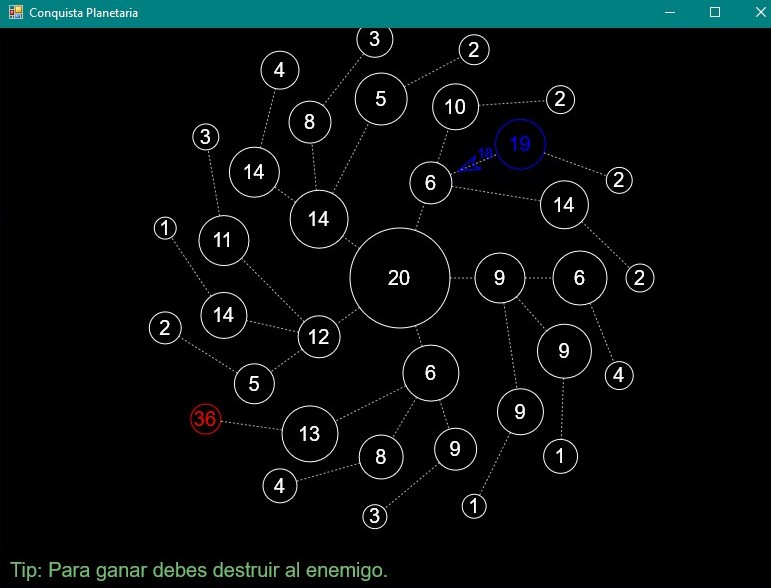
Los errores con los que me tope en Consulta2() no me encontre con mucha dificultad despues de haber terminado el metodo anterior ya que pense que este serialo inverso, donde en vez de almacenar hasta encontrar el planeta de la ia, una vez encontrado dicho planeta recorrer sus hijos y almacenarlos.

Los errores con los que me tope en Consulta3() fue que 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.

Para CalcularMovimiento() tuve mucho trabajo tratando de identificar los casos excepcionales pero solo me llevo tiempo de analicis y pruebas. Tambien me paso con la formula de pedir ayuda que tube que cambiarla por algo mas complejo y firme

# Capturas del Juego



Como se verá en las imágenes anteriores, se muestra como se desarrolla el movimiento del planeta de la ia a medida que avanza el juego.

La ia esta programada para que en el caso de tener mayor población que el siguiente planeta ataque y sino pedir ayuda de su antecesor y así hasta poder atacar.

Como verán la ia avanza sin problemas desde una posición apartada hacia el planeta del jugador.

# Sugerencias de Mejora

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 a la hora de armar los diferentes métodos, pero pudo completarse con rango de margen de tiempo.

Se tomaron en consideración muchas problemáticas a la hora de plantear el movimiento. Al principio se idearon varias cosas, pero al ver como funcionaban y su cantidad de código fueron quitadas. Por este motivo se llevaron muchas pruebas hasta quedar satisfecho con el resultado final.