UNIVERSIDAD NACIONAL ARTURO JAURETCHE

INSTITUTO DE INGENIERÍA Y AGRONOMÍA

Complejidad Temporal, Estructura de Datos y Algoritmos

INFORME DE TRABAJO PRÁCTICO FINAL

Profesor: Amet Leonardo

2° cuatrimestre de 2021

Alumna*: García Giménez Camila*
DNI: 4009473
Legajo: *55401*

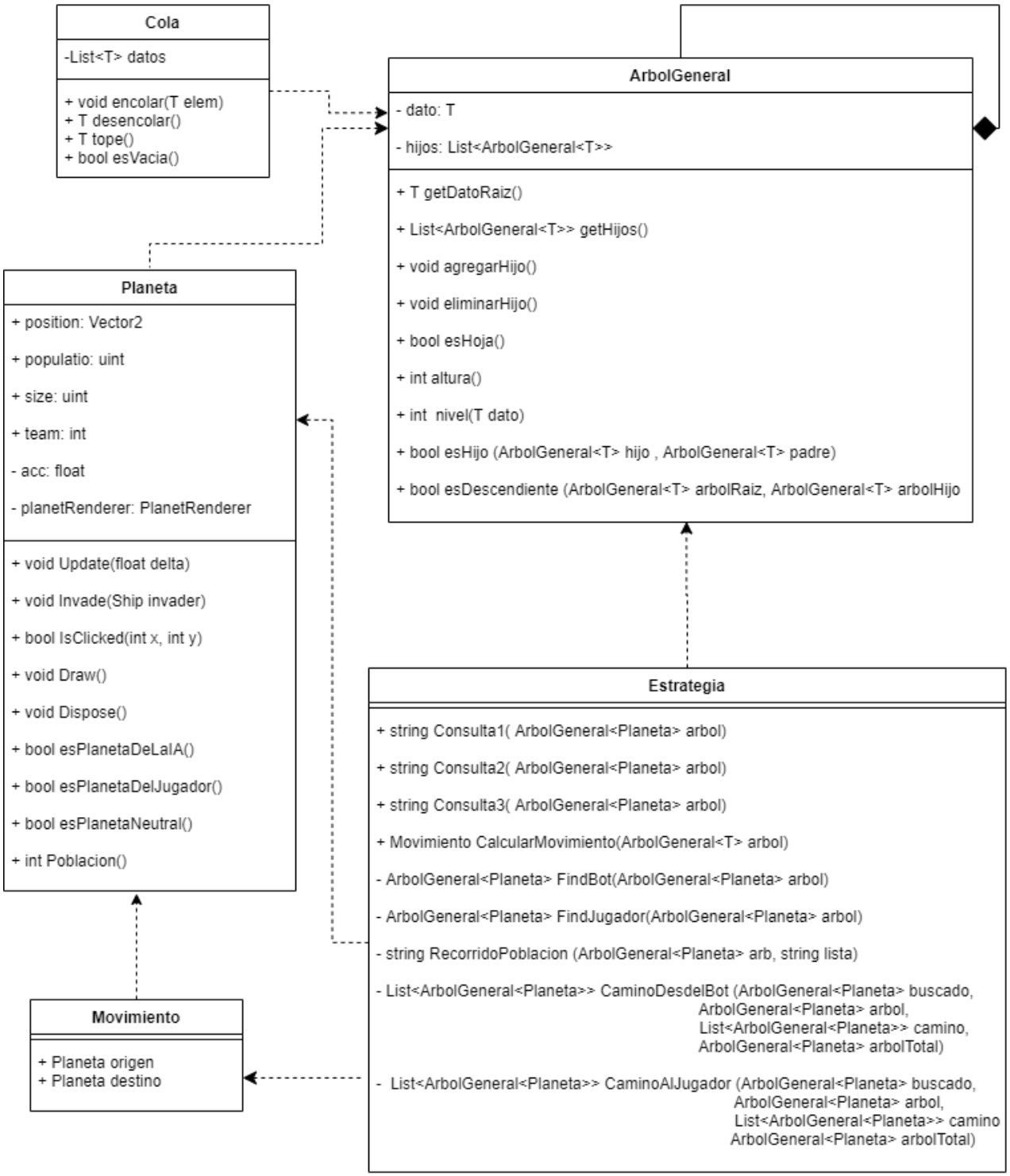
camilavgarcia.gimenez@gmail.com

Introducción

En el presente se describen las implementaciones llevadas a cabo sobre la estructura de juego proporcionada por la cátedra de Complejidad Temporal, Estructura de Datos y Algoritmos. Para ello, se presenta el UML de las clases sobre las que se trabajó, luego se muestran detalles de la implementación: qué se buscaba y cómo se llevó a cabo; como así también comentarios sobre el proceso mismo de implementación e ideas que permitirían una mejora en la funcionalidad. Finalmente, se agrega una conclusión a modo de reflexión acerca del proceso de realización de este trabajo.

Descripción de la estructura del sistema

Figura A. Diagrama UML de las clases utilizadas



Descripción y detalles de implementación

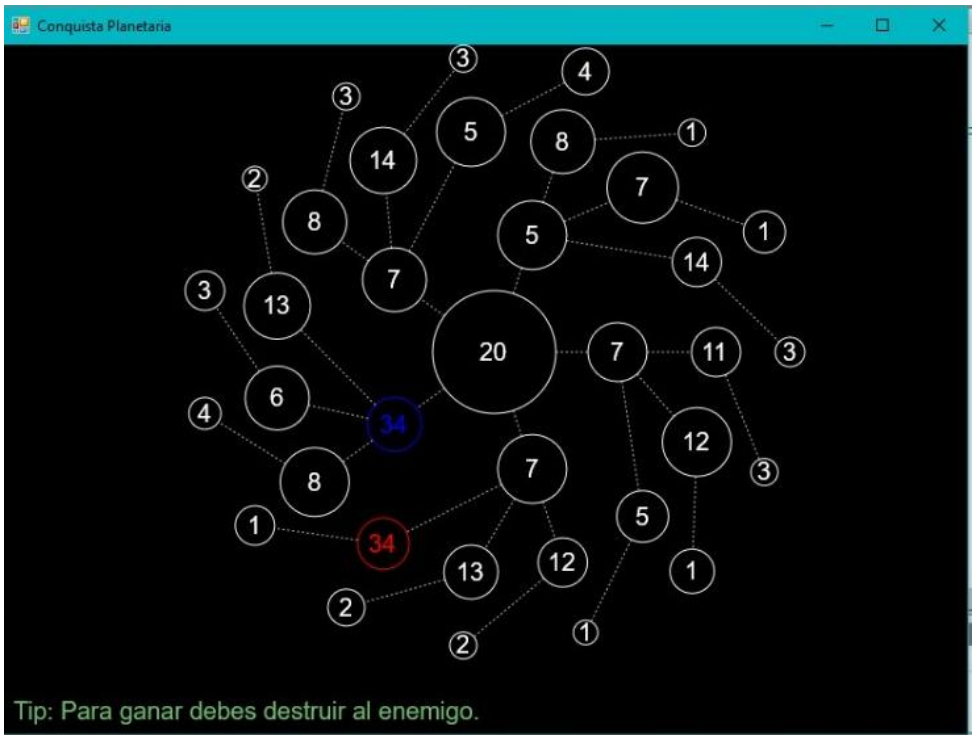
Todas las implementaciones y pruebas que se nombran y muestran en este informe fueron hechas sobre SharpDevelop.
El objetivo del trabajo fue codificar métodos que permitieran, por un lado, realizar consultas en el menú del juego, y por otro, llevar a cabo la estrategia de movimiento del Bot. Todos ellos se crearon dentro de la clase Estrategia, y debían cumplir con determinados objetivos.

"Consulta1"

El objetivo de esta consulta es retornar un texto en el que se indique la distancia del camino desde la raíz hasta el nodo que contiene al planeta del Bot.

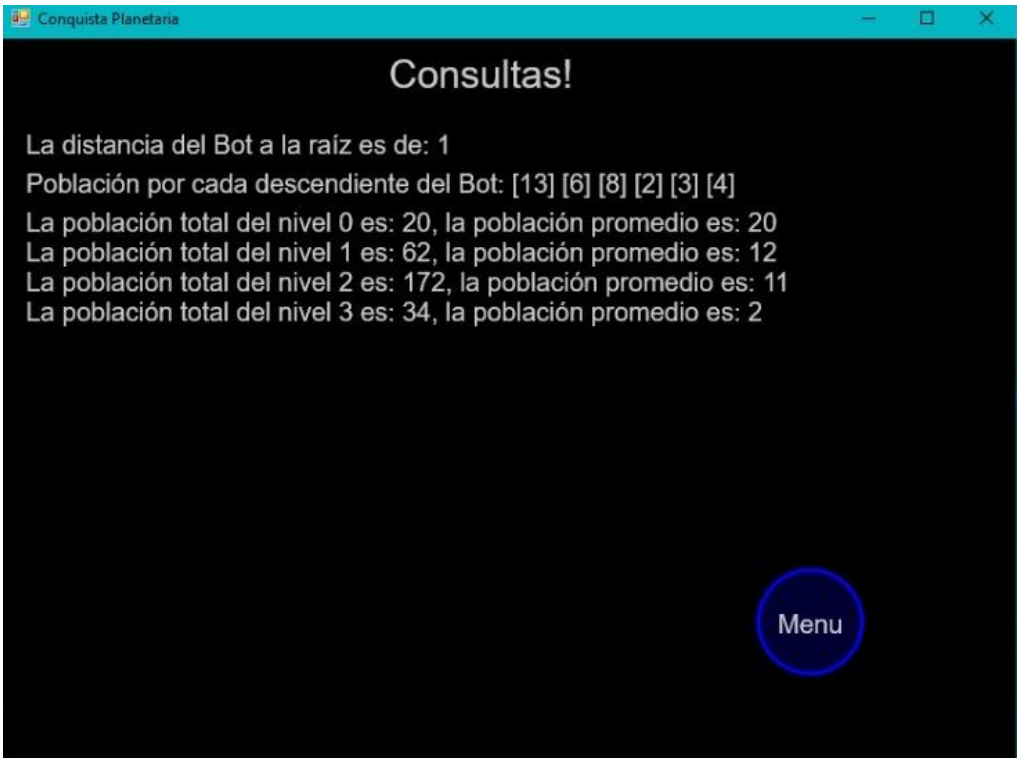
Esta consulta fue implementada utilizando, una llamada al método "FindBot" de la misma clase (Estrategia), el cual busca el nodo perteneciente al Bot a través de un recorrido por niveles con separación. A su vez, tras este retornar el objeto "bot", se llama al método "niveles" de la clase "ArbolGeneral", lo que finalmente nos da (como tipo int) la distancia del Bot a la raíz. Esta consulta no presentó inconvenientes o dificultades. Sin embargo, en un primer momento se implementó haciendo todo el código de recorrido por niveles dentro del mismo método, el cual retornaba un dato de tipo int al encontrar ara encon dentro del mismo método, y luego, se consideró conveniente para futuras utilidades que la clase "ArbolGeneral" tuviese un método que retorne el nivel de un objeto dado (pasado como parámetro). A su vez, en la misma línea, es que se pensó e implementó el método privado FindBot para reutilización de código y uso de encapsulamiento.

Imagen 1. Captura de pantalla del juego en ejecución.



Como se ve en la Imagen 2, nos permite saber la distancia del nodo del Bot a la raíz, indicando en pantalla “La distancia del Bot a la raíz es de: ”

Imagen 2. Captura de pantalla de la opción “Consultas” correspondiente a la ejecución juego de la Imagen 1.



“Consulta2”

Esta consulta tiene como objetivo retornar un texto en el que se indiquen todos los planetas descendientes del nodo que contiene al planeta del Bot. Dado que el dato que se tiene sobre cada planeta es su población, se imprime esta misma.

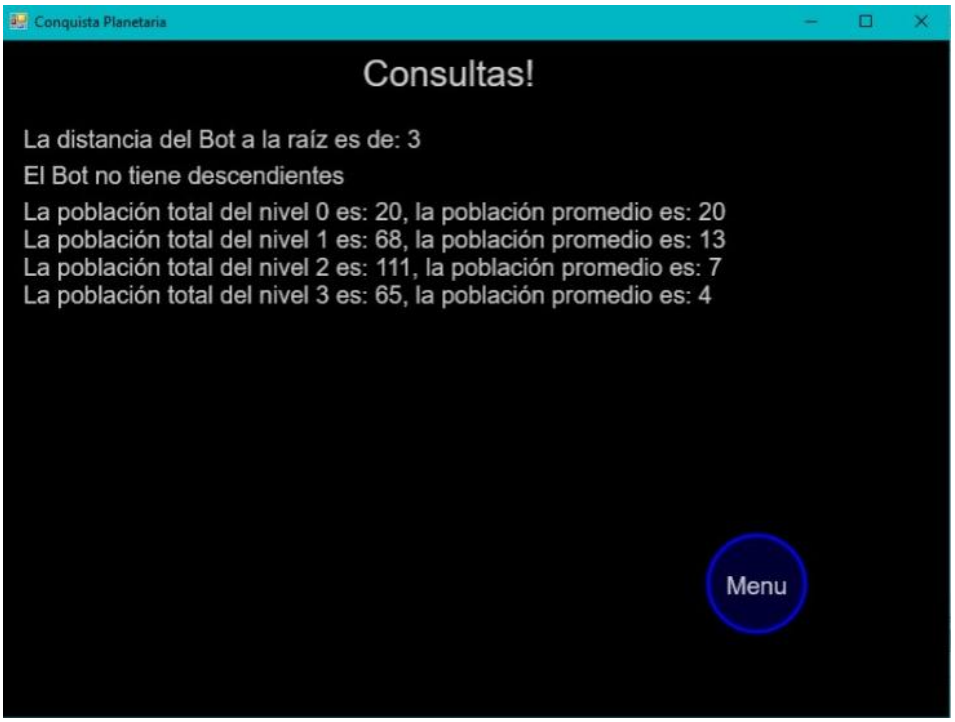
Este método fue implementado utilizando una llamada al método privado “FindBot” para encontrar el Bot, y el método privado “RecorridoPoblacion” que recorre, utilizando una Cola (recorrido por niveles), los descendientes del nodo Bot (objeto ArbolGeneral<Planeta> pasado como parámetro) tomando de cada uno de ellos la población a través del método de la clase Planeta: “Poblacion”, y agregando esta información en un string (también pasado como parámetro).

Esta consulta no presentó dificultad, salvo una “mínima” por la condición de que debía retornar un string, la cual hizo pensar un modo de ir conectando esta información de cada descendiente y retornarla en un string.

Como se ve en la Imagen 3, cuando el Bot tiene planetas debajo de él, esta consulta indica en pantalla “La población por cada descendiente del Bot es:” y seguidamente muestra el dato de la población de cada uno de ellos.

En cambio, como se muestra en la Imagen 4, cuando el nodo al que pertenece el planeta Bot no tiene “hijos”, esta consulta indica en pantalla “El Bot no tiene descendientes”.

Imagen 3. Captura de pantalla de la opción Consultas con juego en ejecución.



“Consulta3”

En este método el objetivo es calcular y retornar la población promedio y total por cada nivel del árbol.

Esto fue llevado a cabo utilizando un recorrido por niveles con separación, en el que por cada nivel que se va separando (por medio de un null), se agrega en el string a retornar, otro string con la información calculada hasta ese momento, y luego, para poder calcular correctamente en cada nivel, se les asigna nuevamente el valor 0 a cada variable utilizada para el cálculo de total, promedio y cantidad de hijos por nivel.

Este método “Consulta3” es el que permite ver en la pantalla de opciones la información de la población por cada nivel tal como se muestra en las imágenes 2 y 3.

“CalcularMovimiento”

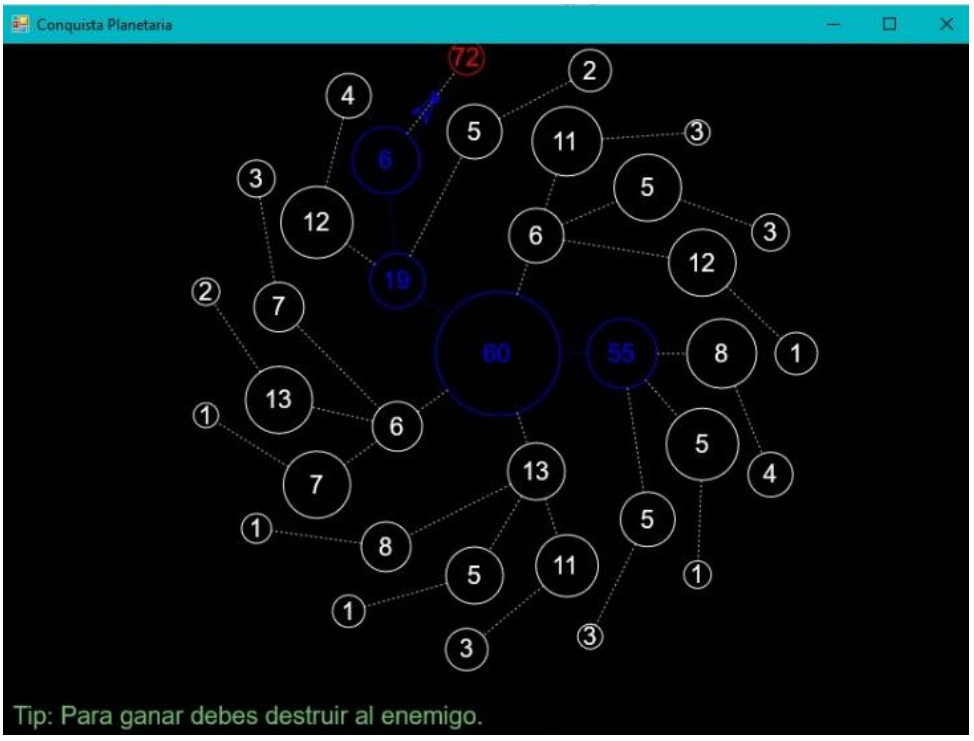
Este método retorna el movimiento apropiado del Bot, según el estado del juego, para que este ataque el planeta enemigo.

Esto fue implementado utilizando, en primer lugar, los métodos privados “FindBot” y “FindJugador” para obtener los objetos pertenecientes a cada uno (es decir, al planeta perteneciente al Bot y al perteneciente al Jugador-humano). Luego, se crean listas dinámicas de ArbolGeneral<Planeta> que se corresponde con el recorrido de ataque que debería seguir, dependiendo el estado del juego, el planeta del Bot. Así mismo, se creó un método privado que es utilizado para evaluar, de cierta manera, el estado del juego. Este método “esDescendiente” retorna *true* si un objeto pasado como parámetro es descendiente de otro también pasado como parámetro, y fue utilizado de manera tal que resulta importante en el establecimiento de los movimientos a realizar por el Bot.

Para evaluar la situación del juego, se establece un condicional, en el que si el planeta del jugador es descendiente del planeta del Bot, se ejecuta el código dentro de este. De lo contrario, se ejecuta el código que corresponde al cálculo del camino -esto es, el método “CaminoDesdeBot”- que debería seguir el Bot para poder llegar (por “default”) a la raíz, y seguidamente se retorna el movimiento que tiene como origen y destino a los elementos dentro de la lista enlazada “caminoDesdeBot”.

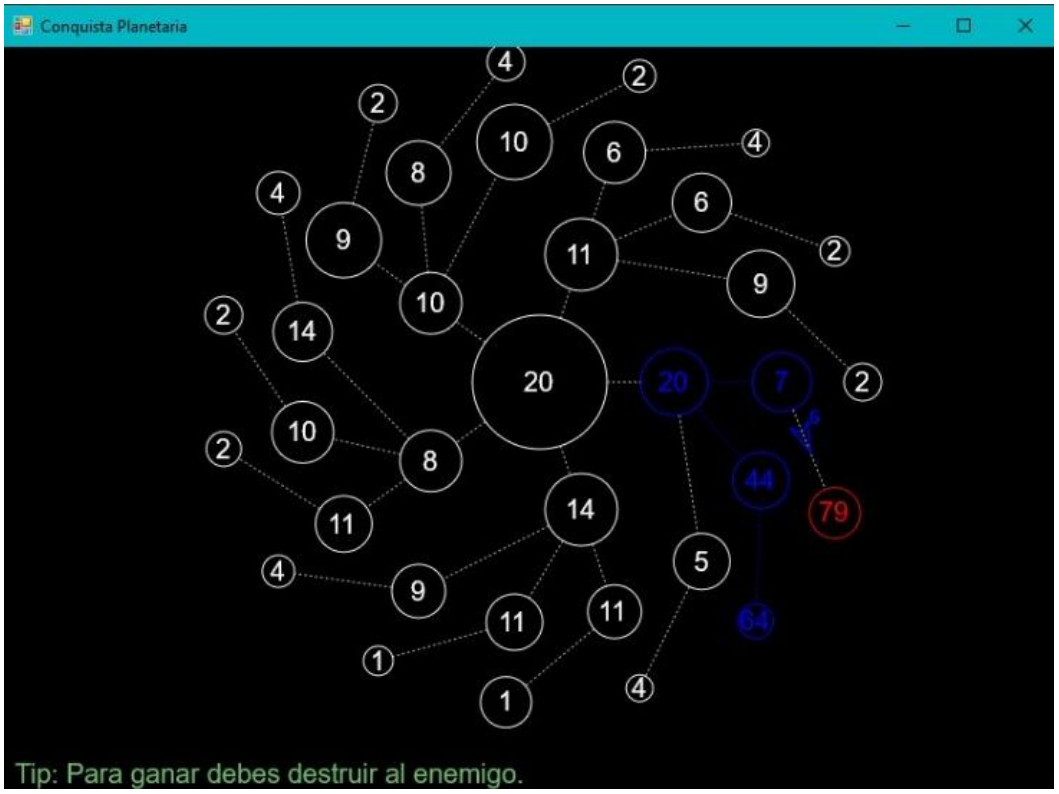
En caso de resultar verdadero el primer condicional (Jugador descendiente del Bot) -e independientemente de si ya se ejecutaron los primeros movimientos o no-, se evalúa si el planeta Jugador es hijo directo del planeta Bot, si esto es así, se retorna un movimiento con el Bot como origen y el Jugador como destino. En caso de que este no sea hijo directo (solo descendiente), se completa la lista enlazada “caminoHaciaJugador” a través del método privado “CaminoHaciaJugador” y, se retorna, por cada elemento, un movimiento que tiene como origen al último planeta de la IA evaluado de la lista, y como destino, al elemento siguiente.

Imagen 4. Captura de pantalla. Bot llega al jugador a través de la raíz.



Este método fue el que presentó mayor dificultad al momento de intentar implementarlo. Primero puede mencionarse la dificultad al intentar que el Bot continúe moviéndose aun cuando la raíz ya era perteneciente a la IA. Más precisamente: el Bot atacaba al Jugador siempre y cuando este se encontrara a distancia 1 de la raíz y no a más (salvo en un intento, en el que "mejoró" a 2 de distancia). Esta situación llevó a muchos intentos fallidos de solución, y para que puedan llevarse a cabo correctamente los movimientos necesarios, se cambió la estructura general del código del método hecha hasta ese momento. La estructura del código tal como está, permite, ahora sí, que se retornen movimientos correctamente sea cual sea el nivel del planeta del jugador.
Por otra parte, también puede indicarse que la implementación del método, tal como está presentada, se redujo a dos condiciones generales básicas (el jugador es descendiente o no lo es), y a partir de ahí se aplican otros condicionales que ya mencionaron más arriba. De esta manera, y tras, cortar partes redundantes e innecesarias, y modificar parte del código, se logró reducirlo de manera que en menos líneas, pueda ejecutar lo que anteriormente.

Imagen 5. Captura de pantalla. Bot llega a Jugador solo atravesando al primer nodo ancestro del planeta del jugador.



Ideas y sugerencias para la implementación de mejoras

Si bien las implementaciones permiten una funcionalidad adecuada, se buscó cierta eficiencia (o por lo menos no una sobrecarga innecesaria) y se reutilizó código y ocultó/encapsuló lo que se consideraba adecuado, sin dudas pueden implementarse muchas mejoras. Partiendo de la optimización del código y utilización de buenas prácticas, hasta (y sobre todo) la estrategia de movimientos del Bot para atacar e intentar destruir al enemigo. Específicamente en el método "CalcularMovimiento" es que podrían implementarse condiciones en las que al ya tener los nodos que corresponden a la lista de camino pertenecientes a la IA, estos se recarguen ya desde el primer tramo (comenzando desde el nodo inicial a la raíz o ancestro en común). Para luego, ya sí en el último tramo, es decir, desde la raíz/ancestro en común hasta el jugador (esto último si se ha logrado) y seguir atacando pero ya con muchas más naves y hacer así más efectiva la derrota al jugador humano. Así mismo, podría establecerse una condición más específica en la que se recarguen desde determinado nodo, dependiendo de la Población del planeta del jugador, evaluando primero cuánto tarda en realizarse el envío de naves y cuántas se cargan automáticamente en un período determinado.

Conclusiones

La realización de este Trabajo Final constituyó un desafío que resultó muy interesante. A la vez que permitió y llevó a poner en práctica algunos de los algoritmos vistos durante la cursada de la materia, y a tener consideración y cuidado con respecto a los tiempos de ejecución, y por ende, a la eficiencia de cada parte de código, ya no sólo el mero hecho de hacerlo.
Personalmente, aún pese al cierre de la materia, este trabajo me motiva a seguir buscando soluciones y estrategias diferentes. Además de las posibilidades en torno a mejorar/optimizar el código, la dinámica del juego permite jugar -valga la redundancia- con las estrategias posibles de movimiento. Y en mi opinión es eso lo que lo hace doblemente interesante (y desafiante).
En síntesis, considero que la experiencia de la realización del trabajo, más allá de la experiencia personal, es sumamente enriquecedora.