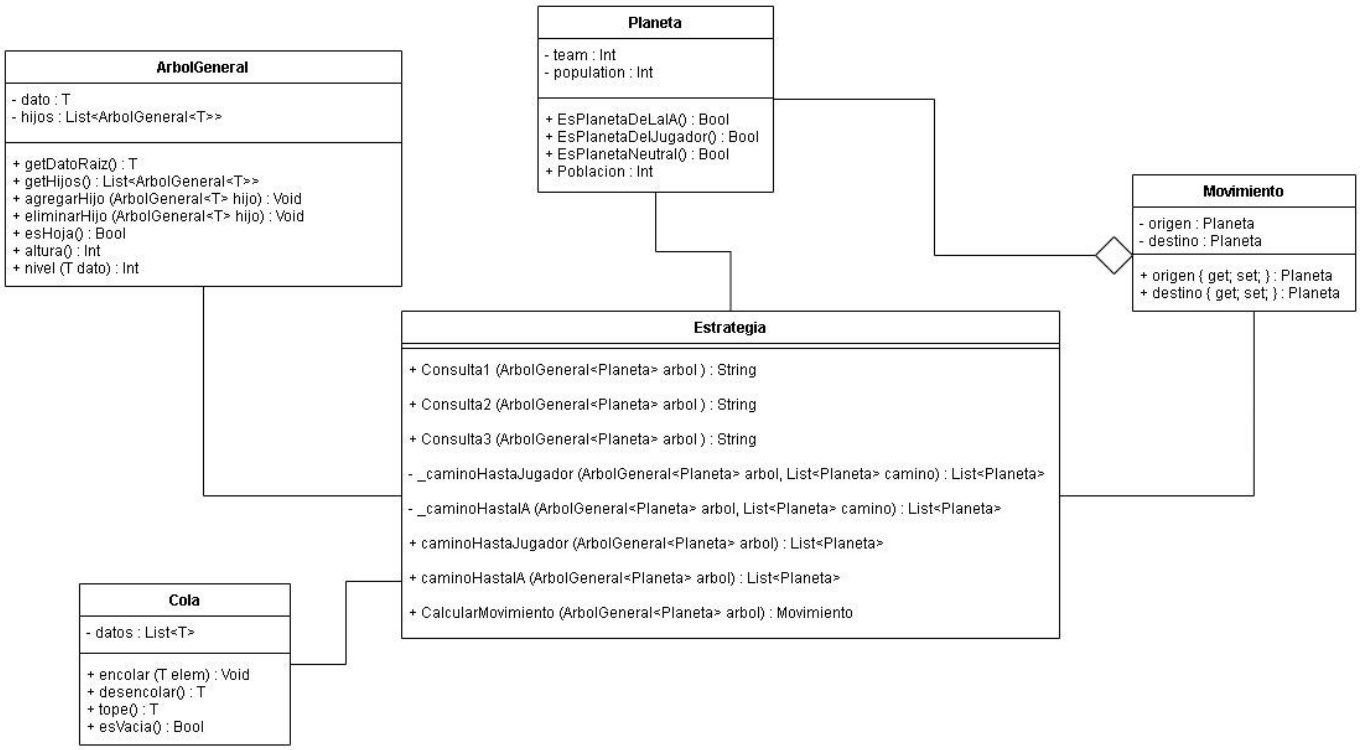
| | | |
|---|---|---|
| Universidad Nacional ARTURO JAURETCHE | **Complejidad Temporal, Estructuras de Datos y Algoritmos**<br><br>**Trabajo Práctico Final** | **Docente:** Leonardo J. Amet<br><br>**Alumno:** Delgado, Javier<br><br>**Comisión:** 5 |

## Introducción

Se desarrolló y codifico una implementación para el funcionamiento estratégico de un juego solicitado, el cual tiene como objetivo principal, brindar un alto grado de dificultad a sus jugadores.

## Metodología




En primer lugar, se codificaron las consultas 1,2 y 3, con un recorrido por niveles, utilizando una cola y un separador de nivel, la metodología para los 3 casos fue muy similar:

- Consulta 1: Se agregó una variable para guardar el nivel en cada recorrido
- Consulta 2: Se agregó una variable para guardar la cantidad de planetas.
- Consulta 3: Se agregó una variable para guardar la población de cada planeta

```
public String Consulta1( ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>(); //Inicializamos la cola
    ArbolGeneral<Planeta> aux;  //Declaramos la variable auxiliar
    int nivel = 0;              //En esta variable guardamos el nivel en donde nos encontremos
    c.encolar(arbol);           //Encolamos el arbol
    c.encolar(null);            //Encolamos el separador
    while(!c.esVacia())
    {
        aux = c.desencolar();
        if(aux != null)
        {
            if(aux.getDatoRaiz() != null)
            {
                if(aux.getDatoRaiz().EsPlanetaDeLaIA()) //Si el planeta encontrado pertenece a la IA, retornamos el nivel actual
                {
                    return "Distancia de raiz al planeta mas cercano(bot): "+nivel;
                }
            }
        }
        if(aux == null) // En caso de que encontramos el separador, aumentamos el nivel
        {
            nivel++;
            if(!c.esVacia())
            {
                c.encolar(null);
            }
        }
        else    // Encolamos el resto de los hijos
        {
            foreach (var hijo in aux.getHijos())
            {
                c.encolar(hijo);
            }
        }
    }
    return null;
}
```

```
public String Consulta2( ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> aux;
    int nivel = 0;
    int cantidadPlanetas = 0;   //Guardamos la cantidad de planetas por nivel, con población mayor a 10
    string consulta2 = null;
    c.encolar(arbol);
    c.encolar(null);
    while(!c.esVacia())
    {
        aux = c.desencolar();
        if(aux != null)
        {
            if(aux.getDatoRaiz() != null)
            {
                if(aux.getDatoRaiz().Poblacion() > 10) //Si la poblacion supera las 10 unidades, aumentamos la cantidad de planetas
                {
                    cantidadPlanetas++;
                }
            }
        }
        if(aux == null) //Si encontramos el separador, agregamos la información obtenida del nivel, y reiniciamos las variables.
        {
            consulta2+= "Nivel: "+nivel+" Planetas: "+cantidadPlanetas+" ";
            nivel++;
            cantidadPlanetas = 0;
            if(!c.esVacia())
            {
                c.encolar(null);
            }
        }
        else
        {
            foreach (var hijo in aux.getHijos())
            {
                c.encolar(hijo);
            }
```

```
public String Consulta3( ArbolGeneral<Planeta> arbol)
{
    Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
    ArbolGeneral<Planeta> aux;
    int nivel = 0;
    int cantidadPlanetas = 0;   //Guardamos la cantidad de planetas por nivel
    int poblacion = 0;          //Guardamos la poblacion de cada planeta
    string consulta3 = null;
    c.encolar(arbol);
    c.encolar(null);
    while(!c.esVacia())
    {
        aux = c.desencolar();

        if(aux != null)
        {
            if(aux.getDatoRaiz() != null)   //Acumulamos la suma de poblaciones y aumentamos la cantidad de planetas
            {
                poblacion += aux.getDatoRaiz().Poblacion();
                cantidadPlanetas++;
            }
        }
        if(aux == null) //Finalizando el recorrido del nivel, sacamos el promedio, guardamos, y reinciamos las variables
        {
            float promedioPorNivel = (float)poblacion/cantidadPlanetas;
            consulta3+= "\nNivel :"+nivel+" Promedio poblacional: "+promedioPorNivel;
            nivel++;
            cantidadPlanetas = 0;
            poblacion = 0;
            if(!c.esVacia())
            {
                c.encolar(null);
            }
        }
        else
        {
            foreach (var hijo in aux.getHijos())
            {
                c.encolar(hijo);
```

En segundo lugar, se implementaron 2 métodos para obtener el camino hasta la IA y otro hacia el jugador. Estos 2 métodos constituyen la estrategia principal de la IA, la cual consiste en conquistar primero la raíz del árbol y luego atacar al jugador sucesivamente.

- **caminoHastaIA:** Este método devuelve un camino hasta la posición donde se encuentre la IA actualmente.

- **caminoHastaJugador:** Este método devuelve un camino hasta la posición donde se encuentre el jugador actualmente.

El proceso de ambos métodos es el mismo, lo único que cambia es la condición de corte, es decir, si es planeta de la IA o del jugador.

```
public List<Planeta> caminoHastaIA(ArbolGeneral<Planeta> arbol)
{
    List<Planeta> camino = new List<Planeta>();
    return _caminoHastaIA(arbol,camino);
}

private List<Planeta> _caminoHastaIA(ArbolGeneral<Planeta> arbol, List<Planeta> camino)
{
    //Primero raiz...
    camino.Add(arbol.getDatoRaiz());

    //Si encontramos camino...
    if(arbol.getDatoRaiz().EsPlanetaDeLaIA())
    {
        return camino;
    }
    else
    {
        //Hijos recursivamente...
        foreach(var hijo in arbol.getHijos())
        {
            List<Planeta> caminoAux = _caminoHastaIA(hijo, camino);
            if(caminoAux != null)
            {
                return caminoAux;
            }
        }
        //Saco ultimo planeta de camino
        camino.RemoveAt(camino.Count-1);
    }
    return null;
}
```

```
public List<Planeta> caminoHastaJugador(ArbolGeneral<Planeta> arbol)
{
    List<Planeta> camino = new List<Planeta>();
    return _caminoHastaJugador(arbol,camino);
}

private List<Planeta> _caminoHastaJugador(ArbolGeneral<Planeta> arbol, List<Planeta> camino)
{
    camino.Add(arbol.getDatoRaiz());
    if(arbol.getDatoRaiz().EsPlanetaDelJugador()) //Condicion que cambia con respecto al metodo anterior
    {
        return camino;
    }
    else
    {
        foreach(var hijo in arbol.getHijos())
        {
            List<Planeta> caminoAux = _caminoHastaJugador(hijo, camino);
            if(caminoAux != null)
            {
                return caminoAux;
            }
        }
        camino.RemoveAt(camino.Count-1);
    }
    return null;
}
```

Por último y para finalizar, se implementó el método principal el cual se encarga de desarrollar la estrategia mencionada anteriormente, devolviendo el movimiento que repetirá la IA cada cierto tiempo.

```
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol)
{
    //Si la raiz no esta ocupada por la IA, entonces formara un camino hasta conquistarla.
    if(!arbol.getDatoRaiz().EsPlanetaDeLaIA())
    {
        List<Planeta> mov = caminoHastaIA(arbol);
        for(int i = mov.Count-1; i > 0; i--)
        {
            if(mov[i].EsPlanetaDeLaIA()) //Los movimientos son desde la posicion donde se encuntra la IA hasta la raiz
            {
                return new Movimiento(mov[i],mov[i-1]);
            }
        }
    }
    /*Una vez conquistada la raiz, la IA solo se dedica a formar caminos hacia el jugador, atacandolo hasta que pierda,
     donde los movimientos ahora serian en sentido opuesto... */
    else
    {
        List<Planeta> movAtaque = caminoHastaJugador(arbol);
        for(int i = 0 ; i < movAtaque.Count-1; i++)
        {
            /*Si la raiz fue conquistada, la primera posicion corresponderia a la IA, por lo cual los movimientos
             siguientes serian una posicion mas adelante*/
            if(!movAtaque[i+1].EsPlanetaDeLaIA())//Los movimientos son desde la raiz hasta el jugador humano
            {
                return new Movimiento(movAtaque[i],movAtaque[i+1]);
            }
        }
    }

    return null;
}
```

Si el jugador llegara a ocupar la raíz que conquisto la IA previamente, la estrategia vuelve a iniciar, por lo cual la IA volvería a conquistar la raíz y posteriormente atacar al jugador.

**Resultados**

Los resultados fueron satisfactorios, pero no los esperados, uno de los inconvenientes que surgió a la hora de desarrollar la estrategia de la IA, es el retardo que había en cada llamada al método **calcularMovimiento()**, el cual era de aproximadamente entre 3 o 4 segundos. Es decir, la efectividad de la estrategia por mas buena que sea, siempre iba ser inferior a cualquier movimiento que pudiera realizar el jugador humano, debido a la gran diferencia de cantidad de movimientos que se pueden realizar en cada lapso de tiempo.

Como posible mejora, se podría reducir el tiempo de llamada al método mencionado anteriormente, permitiendo una vez conquistada la raíz, poder ejecutar ataques simultáneos en todos los planetas cercanos, aumentando considerablemente las probabilidades de ganar de la IA.

Cabe mencionar, que se encontraron algunos problemas menores en cuanto a la interfaz gráfica, posiciones iniciales del juego, etc. Pero todo esto escapa de nuestra tarea específica.

## Conclusión

Para concluir, se pudieron aplicar varios conceptos vistos en la materia, y se pudieron implementar sin inconvenientes, si bien el desarrollo de este trabajo tuvo un cierto grado de dificultad, se proporcionaron todas las herramientas necesarias para su realización. Por lo tanto, el aprendizaje y conocimiento obtenido durante este proceso, fue muy gratificante y alentador.