TRABAJO FINAL DE PROGRAMACIÓN

IMPLEMENTACION DE ESTRATEGIA PARA CONQUISTA PLANETARIA

Autor: Matías Ismael Cruz

Materia: Complejidad temporal

Carrera: Ingeniería en informática

Institución: Universidad Arturo Jauretche

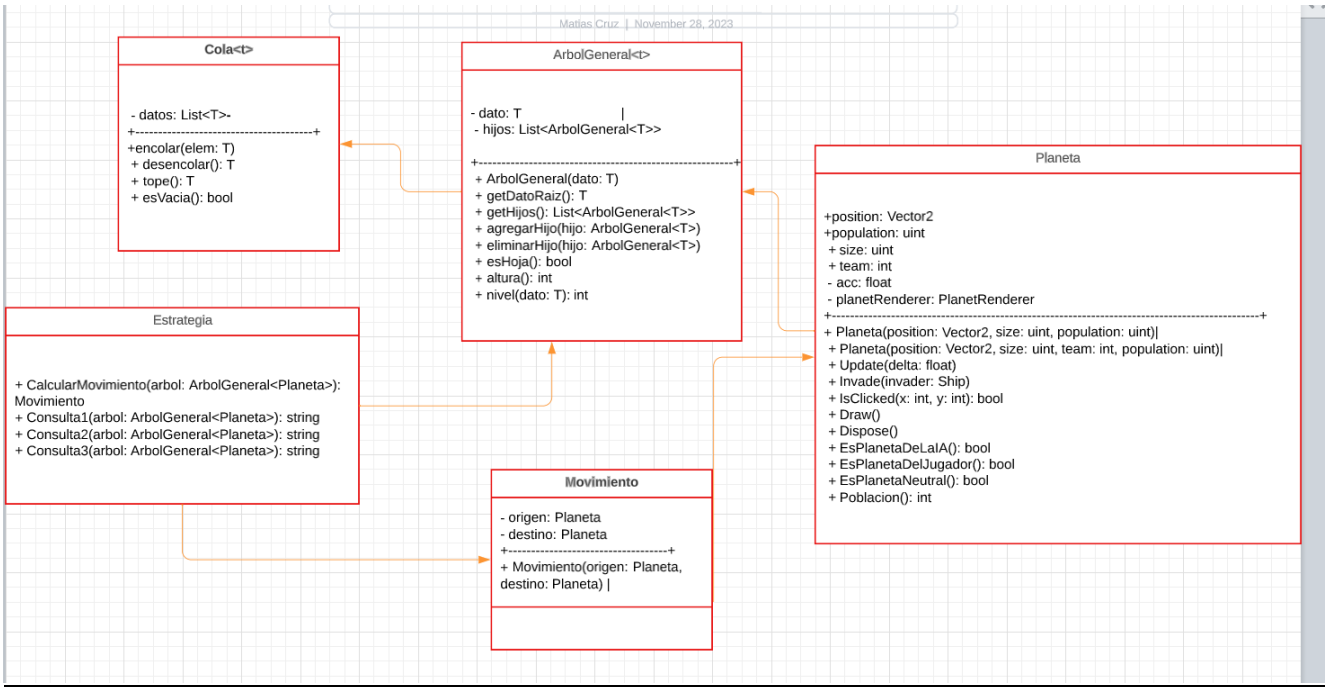
Introducción:

En el ámbito de la programación y el desarrollo de videojuegos, la implementación de estrategias para la conquista planetaria en un entorno virtual representa un desafío emocionante y multifacético. Este trabajo final se centra en la creación de un sistema de juego que simula una contienda planetaria entre un jugador humano y una entidad de software con estrategias predefinidas (Bot). La dinámica del juego se basa en la toma de decisiones estratégicas, la conquista de planetas y la gestión de recursos.

Objetivo:

El objetivo principal de este proyecto es diseñar e implementar una estrategia de ataque para el Bot en el contexto de un juego de conquista planetaria. Para lograr esto, se utiliza un modelo jerárquico de planetas representados mediante un árbol general. La estrategia del Bot se fundamenta en la toma de decisiones calculadas en función de la estructura del árbol planetario.

DIAGRAMA DE CLASES UML



Capturas de pantalla del código implementado junto con su explicación

```
namespace DeepSpace
{

    class Estrategia
    {


        public String Consulta1(ArbolGeneral<Planeta> arbol)
        {
            Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
            ArbolGeneral<Planeta> Arbolaux;

            int distancia = 0;
            cola.encolar(arbol);
            cola.encolar(null);

            while (!cola.esVacia()){
                Arbolaux = cola.desencolar();
                if(Arbolaux == null){
                    distancia++;
                    if(!cola.esVacia()){
                        cola.encolar(null);
                    }
                }
                else{
                    foreach (var hijos in Arbolaux.getHijos()){
                        cola.encolar(hijos);

                    }
                    if(Arbolaux.getDatoRaiz().EsPlanetaDeLaIA()){

                        return "El bot se encuentra a una distancia de " + distancia + " de la raiz";

                    }

                }


            }

            return "algo salio mal";

        }
```

Este fragmento de código implementa una función llamada Consulta1 que realiza un recorrido en amplitud (BFS) en un árbol general de planetas para determinar la distancia desde la raíz del árbol hasta el primer planeta que pertenece al Bot.

Inicialización:

- Se crea una cola Cola<ArbolGeneral<Planeta>> para realizar el recorrido en amplitud.
- Se inicializa una variable Arbolaux para almacenar el nodo actual durante el recorrido.
- Se inicializa una variable distancia para contar la distancia desde la raíz hasta el primer planeta del Bot.
- Se encola el árbol de planetas inicial (arbol) en la cola.
- Se encola un marcador null para indicar el final de un nivel en el árbol.

Recorrido en Amplitud:

- Se inicia un bucle while que continúa hasta que la cola esté vacía.
- Se desencola un elemento de la cola y se almacena en Arbolaux.
- Si Arbolaux es null, significa que se ha completado un nivel en el árbol, por lo que se incrementa la variable distancia y se encola otro null si la cola no está vacía.
- Si Arbolaux no es null, se encolan todos los hijos de Arbolaux en la cola.
- Si el planeta en la raíz del subárbol actual pertenece al Bot (EsPlanetaDeLaIA()), la función devuelve un mensaje indicando la distancia y termina.
- Si el bucle termina y no se encuentra ningún planeta del Bot, la función devuelve un mensaje indicando que algo salió mal.

```
public String Consulta2(ArbolGeneral<Planeta> arbol)
{


    String respuesta = " ";
    ArbolGeneral<Planeta> aux = null;
    Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
    cola.encolar(arbol);  //raiz
    while (!cola.esVacia()) {
        aux = cola.desencolar();  //desencola el primer elemento

        if (aux.getDatoRaiz().EsPlanetaDeLaIA()) {         //caso 1: la raiz es el bot
            if (!aux.esHoja()) {
                foreach (var hijo in aux.getHijos()) {
                    respuesta += " " + preordenMain(hijo, respuesta);
                }
            } else {
                respuesta = "es hoja";
            }

            break;
        } else {
            foreach (var hijo in aux.getHijos()) {
                cola.encolar(hijo);
            }


        }
    }

    return "Descendientes del planeta del bot : " + respuesta;

}

public String preordenMain(ArbolGeneral<Planeta> arbol, string respuesta)
{
    respuesta = (arbol.getDatoRaiz().Poblacion()).ToString();
    List<ArbolGeneral<Planeta>> listaHijos = arbol.getHijos();
    foreach (var hijo in listaHijos) {
        respuesta += " " + preordenMain(hijo, respuesta) + "/";
    }
    return respuesta;
}
```

Este fragmento de código contiene dos funciones, Consulta2 y preordenMain, que trabajan juntas para obtener y devolver información sobre los descendientes del planeta del Bot en un árbol de planetas.

- Se utiliza una cola para realizar un recorrido en amplitud en el árbol.
- Se desencola cada nodo y se verifica si el planeta en ese nodo es el planeta del Bot (EsPlanetaDeLaIA()).
- Si es el planeta del Bot, se llama a la función preordenMain para obtener información sobre sus descendientes.
- Si no es el planeta del Bot, se encolan todos los hijos de ese nodo para seguir explorando.

Función preordenMain:
Esta función realiza un recorrido en preorden (raíz-hijo izquierdo-hijo derecho) en el subárbol dado y devuelve información sobre la población de cada planeta en el subárbol.

```
public String Consulta3(ArbolGeneral<Planeta> arbol)
{
    String respuesta = "";
    int poblacionTotal = 0;
    int Planetaspornivel = 0;
    int nivel = 0;
    int promedio = 0;
    ArbolGeneral<Planeta> aux = null;
    Cola<ArbolGeneral<Planeta>> cola = new Cola<ArbolGeneral<Planeta>>();
    cola.encolar(arbol);  //raiz
    cola.encolar(null);    //separador de nivels
    while (!cola.esVacia()) {
        aux = cola.desencolar();  //desencola el primer elemento

        if (aux != null) {
            Planetaspornivel++;
            poblacionTotal += aux.getDatoRaiz().Poblacion();


            foreach (var hijo in aux.getHijos()) {
                cola.encolar(hijo);

            }
        } else if (!cola.esVacia() || Planetaspornivel != 0) {

            cola.encolar(null);
            promedio = poblacionTotal / Planetaspornivel;

            respuesta += "Nivel " + nivel + " poblacion total " + poblacionTotal + ", promedio " + promedio + "\n";
            poblacionTotal = 0;
            Planetaspornivel = 0;
            nivel++;
        }

    }
    return respuesta;
}
```

Esta función realiza un recorrido en amplitud (BFS) en un árbol de planetas para calcular y devolver información sobre la población total y el promedio de población por nivel en el árbol.
Inicialización:

- Se inicializan variables para calcular la población total, el número de planetas por nivel, el nivel actual y el promedio de población.

- Se utiliza una cola para realizar un recorrido en amplitud en el árbol.

Recorrido en Amplitud:

- Se inicia un bucle while que continúa hasta que la cola esté vacía.
- Se desencola un elemento y se verifica si es null. Si no es null, se procesa el nodo actual:
- Se incrementa el contador de planetas por nivel.
- Se suma la población del planeta actual a la población total.
- Se encolan los hijos del nodo actual.
- Si el elemento desencolado es null:
- Se encola otro null para marcar el final del nivel.
- Se calcula y almacena el promedio de población por nivel.
- Se actualiza la cadena de respuesta con la información del nivel actual.
- Se reinician las variables para el próximo nivel.

Resultado:

- La función devuelve una cadena que contiene información sobre la población total y el promedio de población por nivel en el árbol.

```
public bool caminoJugador(ArbolGeneral<Planeta> arbol, ArbolGeneral<Planeta> origen , List<ArbolGeneral<Planeta>> camino){

    bool caminoEncontrado = false;

    camino.Add(origen);

    if(origen.getDatoRaiz().EsPlanetaDelJugador()){
        caminoEncontrado = true;
    }
    else{
        foreach(var hijo in origen.getHijos()){
            caminoEncontrado = caminoJugador(arbol,hijo,camino);
                if(caminoEncontrado){
                break;
            }
        camino.RemoveAt(camino.Count-1);
        }

    }
    return caminoEncontrado;
```

Esta función “caminoJugador” busca un camino desde un nodo dado en un árbol hasta el primer planeta que pertenece al jugador:

Inicialización:

- Se inicializa una variable caminoEncontrado como false. Esta variable se utiliza para indicar si se ha encontrado un camino desde el nodo actual hasta un planeta del jugador.
- Se agrega el nodo actual al final de la lista camino.

Verificación del Planeta del Jugador:

- Se verifica si el planeta en el nodo actual pertenece al jugador (EsPlanetaDelJugador()).
- Si es así, se actualiza caminoEncontrado a true.

Exploración de los Hijos:

- Si el planeta en el nodo actual no es del jugador, se inicia un bucle foreach para explorar cada hijo del nodo actual de manera recursiva.
- Se actualiza caminoEncontrado con el resultado de la exploración de los hijos.
- Si se encuentra el camino, se sale del bucle.

Retroceso en el Camino:

- Después de explorar los hijos, se retrocede en el camino eliminando el último nodo agregado a la lista camino.

Resultado:

- La función devuelve el valor de caminoEncontrado, indicando si se ha encontrado un camino desde el nodo dado hasta un planeta del jugador.

```
public bool caminoBot(ArbolGeneral<Planeta> arbol, ArbolGeneral<Planeta> origen , List<ArbolGeneral<Planeta>> camino){

    bool caminoEncontrado = false;

    camino.Add(origen);

    if(origen.getDatoRaiz().EsPlanetaDeLaIA()){
        caminoEncontrado = true;
    }
    else{
        foreach(var hijo in origen.getHijos()){
            caminoEncontrado = caminoBot(arbol,hijo,camino);
                if(caminoEncontrado){
                break;
            }
            camino.RemoveAt(camino.Count-1);
        }

    }

    return caminoEncontrado;
}
```

Este método tiene su funcionamiento de forma análoga al anterior.

```
public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol){

     List<ArbolGeneral<Planeta>> jugador = new List<ArbolGeneral<Planeta>>();
     List<ArbolGeneral<Planeta>> bot = new List<ArbolGeneral<Planeta>>();

     //Armo las listas del bot y jugador
     caminoJugador(arbol,arbol,bot);
     caminoBot(arbol,arbol,bot);

     //Evaluo si la lista del bot en la pos 0 no es planeta de la IA, y en ese caso el movimiento es desde el bot hacia el planeta vecino

    if(!bot[0].getDatoRaiz().EsPlanetaDeLaIA()){

        Movimiento movimiento = new Movimiento(bot[bot.Count-1].getDatoRaiz(),bot[bot.Count-2].getDatoRaiz());
        return movimiento;
    }

    int contador = 0;
    //Si el planeta de la pos 0 de la lista de jugador es planeta de la ia, entonces, mientras que el contador sea menor que los elementos de la lista de jugador
    //Entonces, si la lista del jugador en la posicion del contador es planeta de la ia y el planeta vecino no es planeta de la ia, netonces el movimiento
    //seria desde la posicion del contador y la pos del contador mas 1.
    if(jugador[0].getDatoRaiz().EsPlanetaDeLaIA()){
        while(contador < jugador.Count){
            if(jugador[contador].getDatoRaiz().EsPlanetaDeLaIA() && !jugador[contador+1].getDatoRaiz().EsPlanetaDeLaIA()){
                Movimiento movimiento = new Movimiento(jugador[contador].getDatoRaiz(),jugador[contador+1].getDatoRaiz());
                    return movimiento;
            }

            else{
                contador++;
            }
        }
    }

    return null;
}
```

Obtención de los Caminos:

- Se crean listas jugador y bot para almacenar los caminos del jugador y del bot respectivamente.
- Se utilizan las funciones caminoJugador y caminoBot para obtener los caminos correspondientes.

Evaluación del Primer Elemento de la Lista del Bot:

- Se verifica si el primer elemento de la lista del bot no es un planeta de la IA. En ese caso, se genera un movimiento desde el bot hacia el planeta vecino.

Búsqueda del Primer Planeta del Jugador que no es un Planeta de la IA:

- Si el planeta en la posición 0 de la lista del jugador es un planeta de la IA, se busca el primer planeta del jugador que no es un planeta de la IA.
- Se genera un movimiento desde ese planeta hacia el siguiente planeta en el camino del jugador.

Devolución del Movimiento:

- Si se encuentra un movimiento válido, se crea un objeto Movimiento y se devuelve.
- Si no se encuentra un movimiento válido, se devuelve null.

```
        public ArbolGeneral<Planeta> BuscarplanetaIA (ArbolGeneral<Planeta> planeta){

            if(planeta.getDatoRaiz().EsPlanetaDeLaIA()){
                return planeta;
            }
            foreach(var hijo in planeta.getHijos()){
                ArbolGeneral<Planeta> resultado = BuscarplanetaIA(hijo);
                if (resultado != null){
                    return resultado;
                }

            }
            return null;
        }
        public ArbolGeneral<Planeta> BuscarplanetaUsuario(ArbolGeneral<Planeta> planeta){

            if(planeta.getDatoRaiz().EsPlanetaDelJugador()){
                return planeta;
            }
            foreach(var hijo in planeta.getHijos()){
                ArbolGeneral<Planeta> resultado = BuscarplanetaUsuario(hijo);
                if (resultado != null){
                    return resultado;
                }

            }
            return null;
        }
}
```

```
    public ArbolGeneral<Planeta> BuscarplanetaIA (ArbolGeneral<Planeta> planeta){

        if(planeta.getDatoRaiz().EsPlanetaDeLaIA()){
            return planeta;
        }
        foreach(var hijo in planeta.getHijos()){
            ArbolGeneral<Planeta> resultado = BuscarplanetaIA(hijo);
            if (resultado != null){
                return resultado;
            }

        }
        return null;
    }
```

Ambos métodos siguen una estructura similar:

Verificación del Nodo Actual:

- Se verifica si el planeta en el nodo actual cumple con la condición deseada (ya sea ser un planeta de la IA o del jugador).

Recorrido de los Hijos:

- Se realiza un recorrido recursivo de los hijos del nodo actual.

Devolución del Resultado:

- Si se encuentra un planeta que cumple con la condición en los descendientes, se devuelve ese resultado.
- Si no se encuentra en los descendientes, se devuelve null.

Sugerencias: Para realizar un versión avanzada de esta implementación, sería ideal poder aportar más estrategias de ataque en el bot, ya que con las implementadas hasta el momento corresponderían a un nivel “fácil” del juego, puesto que es fácil ganarle al bot:

Análisis de la Tasa de Crecimiento:

Implementar un algoritmo que analice la tasa de crecimiento de los planetas y seleccione objetivos en función de su potencial de desarrollo a largo plazo.

Adaptabilidad Dinámica:

Desarrollar una estrategia que se adapte dinámicamente a las acciones del jugador, respondiendo de manera inteligente a los movimientos y decisiones del oponente.

Simulación de Escenarios:

Incorporar simulaciones de escenarios posibles para prever y anticipar las consecuencias de las acciones del bot, permitiendo una toma de decisiones más informada.

Aprendizaje Automático:

Explorar técnicas de aprendizaje automático para que el bot mejore su rendimiento a lo largo del tiempo, ajustando su estrategia en base a las experiencias pasadas. (Como se está implementando hoy en día con la aparición de las inteligencias artificiales como chat gpt)

Conclusión:

La realización de este trabajo final ha sido una experiencia enriquecedora que ha proporcionado una oportunidad valiosa para aplicar y consolidar los conocimientos adquiridos en el curso. A lo largo del proyecto, he tenido la oportunidad de enfrentar desafíos prácticos, desde la implementación de algoritmos y estructuras de datos hasta la integración de conceptos de inteligencia artificial en el diseño de estrategias de juego.

La complejidad del juego de conquista planetaria ha requerido un enfoque multidisciplinario, combinando aspectos de programación, algoritmos y diseño de software. La creación de la estrategia del bot ha permitido explorar diferentes niveles de dificultad y ha planteado la posibilidad de mejoras continuas, como la incorporación de estrategias más avanzadas o técnicas de aprendizaje automático.

En retrospectiva, este proyecto ha demostrado la importancia de la planificación y la iteración en el proceso de desarrollo. La resolución de problemas, la toma de decisiones y la capacidad para adaptarse a nuevas circunstancias han sido habilidades clave desarrolladas durante esta experiencia. Además, la colaboración y el intercambio de ideas con compañeros me ha contribuido significativamente el aprendizaje y el enfoque crítico en la resolución de desafíos.