TRABAJO PRACTICO INTEGRADOR

Nombre : Walter Miguel

Apellido: pinto

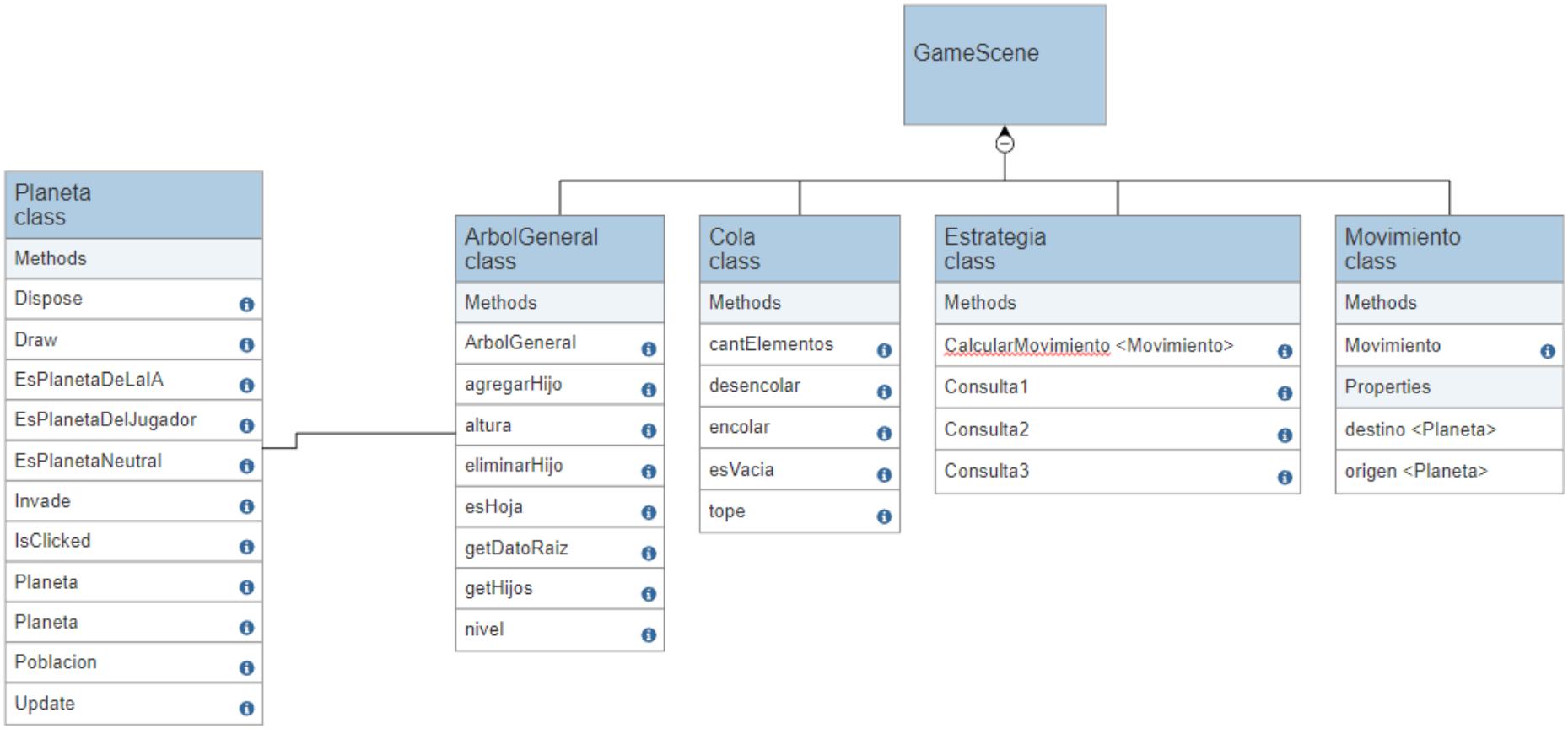
Dni: 36070340

Introducción

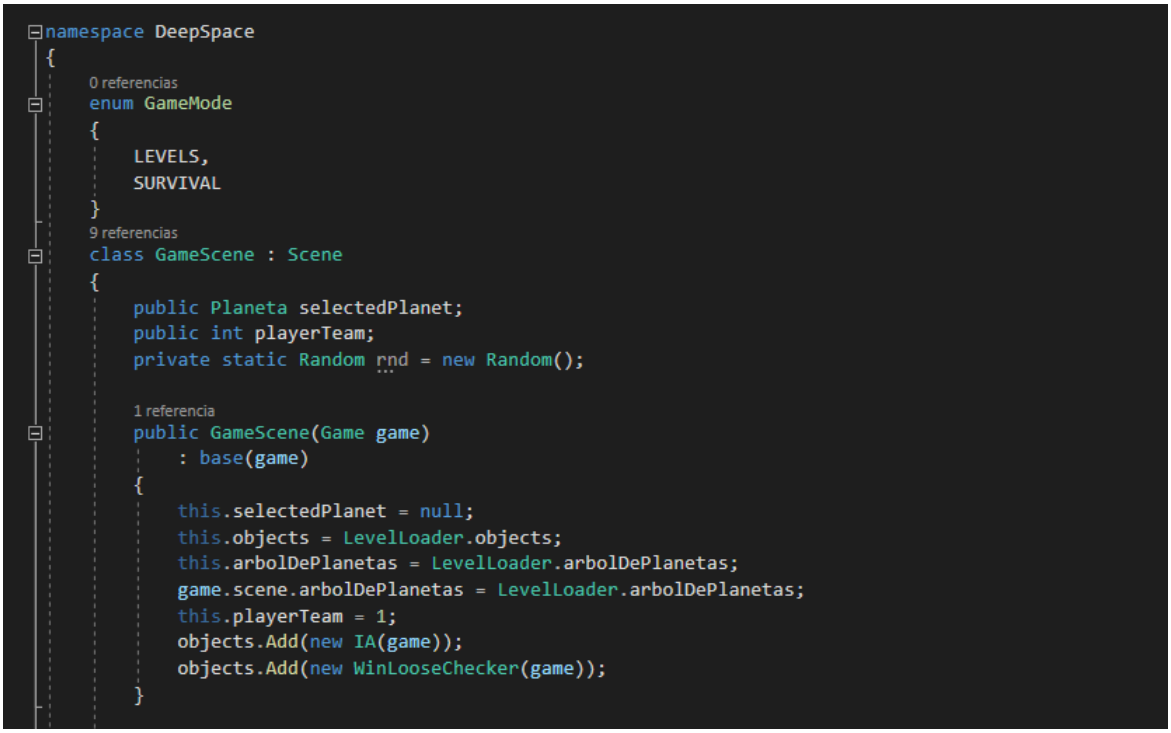
Se presento un programa en el que consiste con un juego de conquistas planetarias, en el cual se me dio a la tarea de programar una estrategia de la IA, la estrategia consistía en detectar el planeta perteneciente a la IA y luego de eso conquistar todos los planetas que están por debajo de este, una vez conquistados todos estos planetas, se debería reagrupar las flotas en el planeta original, enviando las flotas de los planetas conquistados devuelta a este.

Desarrollo

A continuación, se deja el grafico del diagrama uml de las clases principales que componen este programa


GameScene
Planeta class
Methods
Dispose
Draw
EsPlanetaDeLaIA
EsPlanetaDelJugador
EsPlanetaNeutral
Invade
IsClicked
Planeta
Planeta
Poblacion
Update
ArbolGeneral class
Methods
ArbolGeneral
agregarHijo
altura
eliminarHijo
esHoja
getDatoRaiz
getHijos
nivel
Cola class
Methods
cantElementos
desencolar
encolar
esVacia
tope
Estrategia class
Methods
CalcularMovimiento <Movimiento>
Consulta1
Consulta2
Consulta3
Movimiento class
Methods
Movimiento
Properties
destino <Planeta>
origen <Planeta>

El diagrama representa las clases principales que componen al proyecto, en GameScene es donde se instancian las clases.

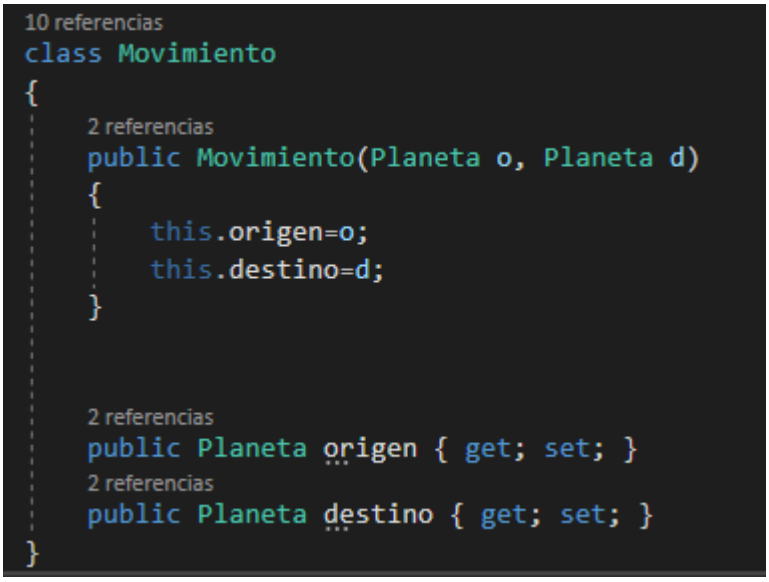


```
namespace DeepSpace
{
    0 referencias
    enum GameMode
    {
        LEVELS,
        SURVIVAL
    }
    9 referencias
    class GameScene : Scene
    {
        public Planeta selectedPlanet;
        public int playerTeam;
        private static Random rnd = new Random();

        1 referencia
        public GameScene(Game game)
            : base(game)
        {
            this.selectedPlanet = null;
            this.objects = LevelLoader.objects;
            this.arbolDePlanetas = LevelLoader.arbolDePlanetas;
            game.scene.arbolDePlanetas = LevelLoader.arbolDePlanetas;
            this.playerTeam = 1;
            objects.Add(new IA(game));
            objects.Add(new WinLooseChecker(game));
        }
```

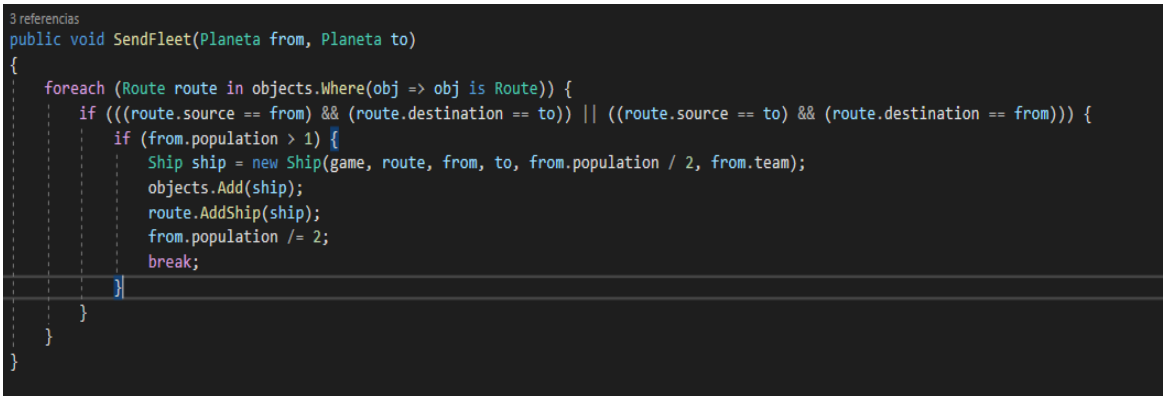
La clase árbol general contiene datos que luego al instanciarse va a ser un planeta y contiene una lista de la misma clase árbol general a la cual se la denomina hijos, además tiene todos los métodos para accederlos

```
33 referencias
public class ArbolGeneral<T>
{

    private T dato;
    private List<ArbolGeneral<T>> hijos = new List<ArbolGeneral<T>>();

    3 referencias
    public ArbolGeneral(T dato) {
        this.dato = dato;
    }

    5 referencias
    public T getDatoRaiz() {
        return this.dato;
    }

    9 referencias
    public List<ArbolGeneral<T>> getHijos() {
        return hijos;
    }

    2 referencias
    public void agregarHijo(ArbolGeneral<T> hijo) {
        this.getHijos().Add(hijo);
    }

    0 referencias
    public void eliminarHijo(ArbolGeneral<T> hijo) {
        this.getHijos().Remove(hijo);
    }

    1 referencia
    public bool esHoja() {
        return this.getHijos().Count == 0;
    }

    0 referencias
    public int altura() {
        return 0;
    }



    0 referencias
    public int nivel(T dato) {
        return 0;
    }
```

movimiento es instanciado cuando el jugador realiza un movimiento en la gameScene y cuando la Ia se instancia, esta instancia a la clase estrategia dando una serie de información con respecto a la ubicación del planeta perteneciente al mismo



Los atributos de movimiento son planeta origen y planeta destino estos mismos se utilizan cuando se instancia en la clase Gamescene mediante un método llamado sendfleet la cual se usa para enviar naves de un planeta a otro



La clase Planeta contiene todos los datos en los cuales van a pertenecer a los nodos del árbol general una vez que se instancien

```
75 referencias
class Planeta : GameObject, IDisposable
{
    public Vector2 position;
    public uint population, size;
    public int team;
    private float acc;
    private PlanetRenderer planetRenderer;

    5 referencias
    public Planeta(Game game, Vector2 position, uint size, uint population)
        : base(game)
    {
        this.position = position;
        this.size = size;
        this.population = population;
        this.acc = 0;
        this.planetRenderer = new PlanetRenderer(this);
    }

    0 referencias
    public Planeta(Game game, Vector2 position, uint size, int team, uint population)
        : base(game)
    {
        this.position = position;
        this.size = size;
        this.population = population;
        this.team = team;
        this.acc = 0;
        this.planetRenderer = new PlanetRenderer(this);
    }

    2 referencias
    public override void Update(float delta)
    {
        acc += delta;
        if (acc > 1 - size/100.0f)
        {
            if (team != 0)
            {
                population += 1;
            }
            acc = acc - 1 + size/100.0f;
        }
    }
```

El método calcular movimiento devuelve un Objeto de tipo movimiento al aplicar una estrategia
solicitada.

```
Cola<ArbolGeneral<Planeta>> q = new Cola<ArbolGeneral<Planeta>>();
q.encolar(arbol);
while (!q.esVacia())
{
    int nivel = q.cantElementos();
    while (nivel-- > 0)
    {
        ArbolGeneral<Planeta> nodo = q.desencolar();
        foreach (ArbolGeneral<Planeta> hijo in nodo.getHijos())
        {
            q.encolar(hijo);
            Planeta planeta = hijo.getDatoRaiz();
            if (planeta.EsPlanetaDeLaIA())
            {
                origen = planeta;

                foreach (ArbolGeneral<Planeta> hijo2 in hijo.getHijos())
                {
                    Planeta planetahijo = hijo2.getDatoRaiz();
                    if (planetahijo.EsPlanetaNeutral() || planetahijo.EsPlanetaDelJugador())
                    {
                        destino = planetahijo;

                        Movimiento nuevaConquista = new Movimiento(origen, destino);
                        Movimiento reagrupar = new Movimiento(destino, origen);
                        conquistas.Add(reagrupar);
                        return nuevaConquista;

                    }
                }
            }
        }
    }
}
```

```
if(contadorConquistas >= conquistas.Count)
{
    contadorConquistas = 0;
}

Movimiento reagrupando = conquistas[contadorConquistas];

contadorConquistas++;
```

Se eligió el recorrido por niveles ya que me pareció mas ordenado conquistar planetas por niveles en vez de en profundidad, además que si se daba el caso de que los hijos del planeta original tuviesen mas hijos la solución por niveles

me pareció la mas adecuada ya que el planeta de origen se iba a ir sustituyendo a medida que cambiaban los niveles.

La primera dificultad la tuve cuando el programa solamente conquistaba en profundidad a pesar de estar utilizando el recorrido por niveles, luego me di cuenta que mi error era no poner un return adentro del recorrido por lo tanto el programa solamente enviaba el ultimo recorrido realizado donde se daba la condición. Luego de poner el return dentro de la condición el programa conquistaba todos los planetas hijos sin ninguna dificultad,

Y lo que más dificultad me trajo fue el reagrupar una vez conquistado todos los hijos, pensé varias maneras pero ninguna funcionaba, poniendo varias condiciones dentro del recorrido lo único que lograba era que no se termine la conquista completa.

Luego se me ocurrió hacer otro recorrido y obtener asi los planetas conquistados y realizar el movimiento en cada uno de ellos, sin embargo no me gusto mucho esta idea, ya que desperdiciaba la información obtenida al momento de conquistar planetas. Por lo que se me ocurrió hacer una lista de movimientos dentro de la clase estrategia, en la cual se iban a ir guardando los movimientos que se hacían al conquistar planetas de manera inversa, luego de eso solo hizo falta un contador para ir recorriendo esa lista a medida que esta se iba ejecutando, el ultimo problema que obtuve fue al momento de que el contador superaba el numero total de movimientos en la lista, y lo que hice fue agregar una condición al final que cuando esto pasara el contador volvería a 0.

Al codificar las consultas solicitadas no hubo muchos inconvenientes, ya que devolver un texto con la información, simplemente consistía en recorrer el árbol y obtener la información y a su vez devolverla dentro de un string.

```
{
    int promedio = promedioArbol(arbol,false);
    int nivelActual = 0;
    int contadorPlanetasMayor = 0;
    string mensaje = "promedio de planetas por nivel \n";
    Cola<ArbolGeneral<Planeta>> q = new Cola<ArbolGeneral<Planeta>>();
    q.encolar(arbol);
    while (!q.esVacia())
    {
        int nivel = q.cantElementos();
        nivelActual++;
        mensaje += "Nivel Actual: "+ Convert.ToString(nivelActual) + "\n";
        contadorPlanetasMayor = 0;
        while (nivel-- > 0)
        {
            ArbolGeneral<Planeta> nodo = q.desencolar();
            foreach (ArbolGeneral<Planeta> hijo in nodo.getHijos())
            {
                q.encolar(hijo);
                Planeta planeta = hijo.getDatoRaiz();
                if (planeta.Poblacion()>promedio)
                {
                    contadorPlanetasMayor++;
                }

            }
        }
        mensaje += "tiene " + Convert.ToString(contadorPlanetasMayor) + "con poblacion mayor al promedio " + "\n";
    }
    return mensaje;
}

2 referencias
private int promedioArbol(ArbolGeneral<Planeta> arbol, bool flag)
{
    Planeta planeta = arbol.getDatoRaiz();
    if (flag)
    {
        return planeta.Poblacion();
    }


    int sum = planeta.Poblacion();
    int cantidadPlanetas = 1;
    foreach (ArbolGeneral<Planeta> nodo in arbol.getHijos())
    {
        sum += promedioArbol(nodo,true);
        cantidadPlanetas++;
    }
    int promedio = sum / cantidadPlanetas;
    return promedio;
}
```

Con la consulta numero 3 que consistía en mostrar por niveles el numero de planetas con población mayor al promedio, lo que se hizo fue dividir el problema en dos partes, primero realice una clase llamada promedioArbol()

donde su objetivo era recorrer todo el árbol y sumar la población de planetas, una vez finalizada se dividía esta suma por la cantidad de planetas y devolvía el resultado. Cuando obtenía el promedio desde la clase consulta3 se realizó un recorrido por niveles en el cual se comparó cada población del planeta con el promedio obtenido anteriormente, sumando así en el contador de planetas con población mayor al promedio cada vez que la comparación así lo demostrara.

A medida que estos datos eran obtenidos se iban guardando en un string en formato de mensaje.

Conclusiones

Al entender como funciona el juego veo la posibilidad de agregarle turnos para que no sea simplemente una carrera de quien clickea más rápido en los planetas.

Con este proyecto pude entender mucho mejor cómo funciona un árbol general y los recorridos de los mismos, a su vez también me ayudo a entender cómo funcionan los demás tipos de arboles que vimos en clase, superar las dificultades que se me iban presentando fue un proceso muy informativo. Ya que pude ver como funcionaba el recorrido a medida que le iba aplicando cambios