

# “DESAFÍO ESPACIAL: IA VS JUGADOR”



Giuliano Mendoza



# Universidad Nacional Arturo Jauretche

27 de Octubre de 2023

Alumno: Giuliano Mendoza

Profesor: Mauro Salina

Materia: Complejidad temporal, Estructuras de datos y Algoritmos.

## Índice

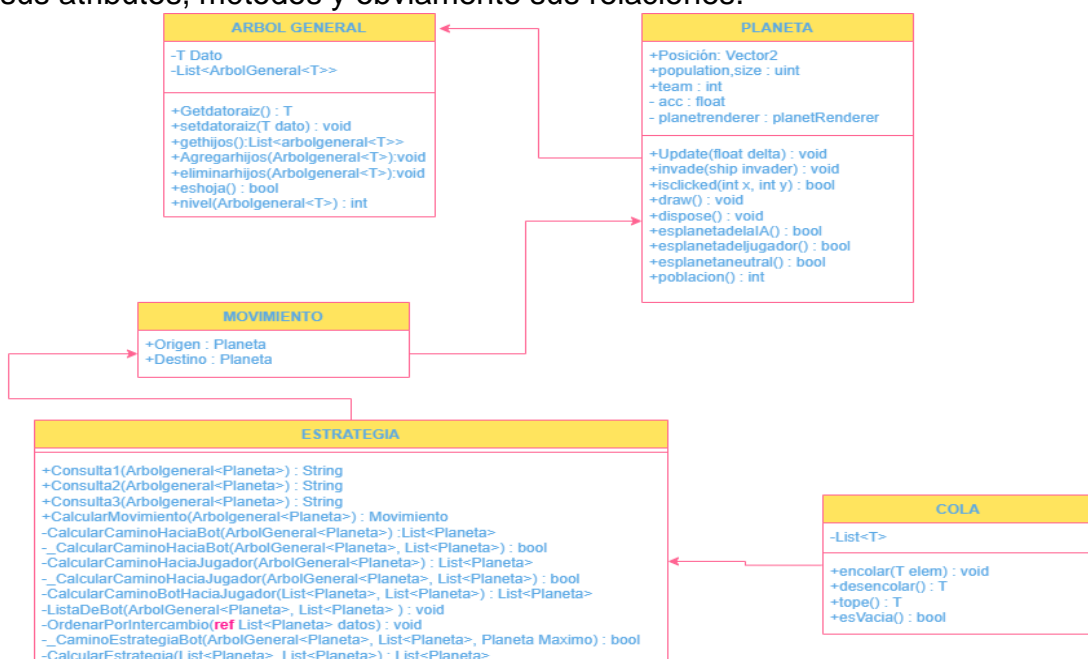
- Introducción
- Diagrama de clases UML
- Detalles de implementación
- Imágenes y descripciones
- Sugerencias
- Conclusión

## Introducción:

Se solicitó el desarrollo de consultas y una estrategia de movimiento para un juego en el cual el objetivo principal era que el Bot siempre ganara, a continuación, se brindara detalladamente la implementación de las consultas y la estrategia de movimiento utilizada para tratar de llegar a ese objetivo.

## Diagrama de clases UML:

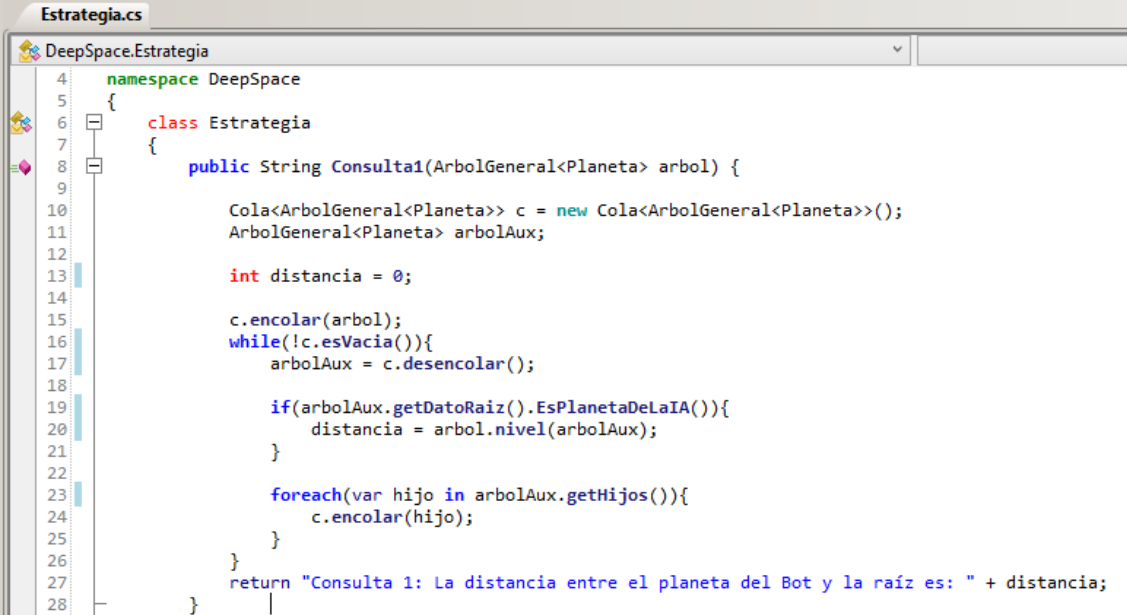
describe la estructura del sistema. El Diagrama muestra las clases del sistema, sus atributos, métodos y obviamente sus relaciones.



## Detalles de implementación:

Durante el desarrollo del trabajo, me enfrente en los momentos donde no sabía para donde arrancar, había leído muchas veces el enunciado tratando de comprender que cosas hacer y que no, escrito hojas llenando de preguntas para poder lograr el funcionamiento de la estrategia. Hasta que, charlando, hablando con compañeros de la cursada pude interpretar bien que hacer y fue momento de meter mano al código.

### Consulta 1:



```
4: namespace DeepSpace
5: {
6:     class Estrategia
7:     {
8:         public String Consultar1(ArbolGeneral<Planeta> arbol) {
9:
10:             Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
11:             ArbolGeneral<Planeta> arbolAux;
12:
13:             int distancia = 0;
14:
15:             c.encolar(arbol);
16:             while(!c.esVacia()){
17:                 arbolAux = c.desencolar();
18:
19:                 if(arbolAux.getDatoRaiz().EsPlanetaDeLaIA()){
20:                     distancia = arbol.nivel(arbolAux);
21:                 }
22:
23:                 foreach(var hijo in arbolAux.getHijos()){
24:                     c.encolar(hijo);
25:                 }
26:             }
27:             return "Consulta 1: La distancia entre el planeta del Bot y la raíz es: " + distancia;
28:         }
```

En este método se crea la clase cola que almacena el árbol pasado como parámetro y un árbol auxiliar que nos servirá para desencolar valores almacenados en la primer cola también utilizo una de distancia 0, para entrar al bucle va a iterar mientras la cola no este vacía, dentro de este bucle se desencola el árbol auxiliar donde se utiliza para comparar si es el planeta del Bot y así agregarle un valor a la distancia utilizando el método “nivel” creado auxiliarmente en la clase de árbol para utilizar este algoritmo sin necesidad de repetirlo ya que lo podríamos llegar a utilizar en otro momento. Caso contrario que no sea planeta del Bot a ese árbol se le pedirá sus árboles hijos que serán almacenados en la cola para luego ser procesados. Luego solo queda retornar un mensaje con la distancia.

### Consulta 2:

```

Estrategia.cs
DeepSpace.Estrategia
28
29
30 public String Consulta2(ArbolGeneral<Planeta> arbol) {
31
32     Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
33     ArbolGeneral<Planeta> arbolAux;
34
35     string lista = "Descendientes del Bot: ";
36     bool existe = false;
37
38     c.encolar(arbol);
39     while(!c.esVacia()){
40
41         arbolAux = c.desencolar();
42
43         if(existe){
44             lista = lista + arbolAux.getDatosRaiz().Poblacion() + " ";
45         }
46
47         if(arbolAux.getDatosRaiz().EsPlanetaDeLaIA()){
48             existe = true;
49             while(!c.esVacia()){
50                 c.desencolar();
51             }
52         }
53         foreach(var hijo in arbolAux.getHijos()){
54             c.encolar(hijo);
55         }
56     }
57     return "Consulta 2: " + lista;
58 }
59

```

Este método tiene una implementación similar al explicado anteriormente, la diferencia que acá no necesitaremos utilizar un separador de niveles ya que lo que busca es encontrar u obtener los planetas descendientes del nodo que contenga el planeta del Bot por lo tanto puede estar directamente en la raíz o puede tener varios descendientes. Por lo que tenemos un String y un booleano, el String para guardar la población de los descendientes en caso de que existan. El booleano para hacer esa comparación.

### Consulta 3:

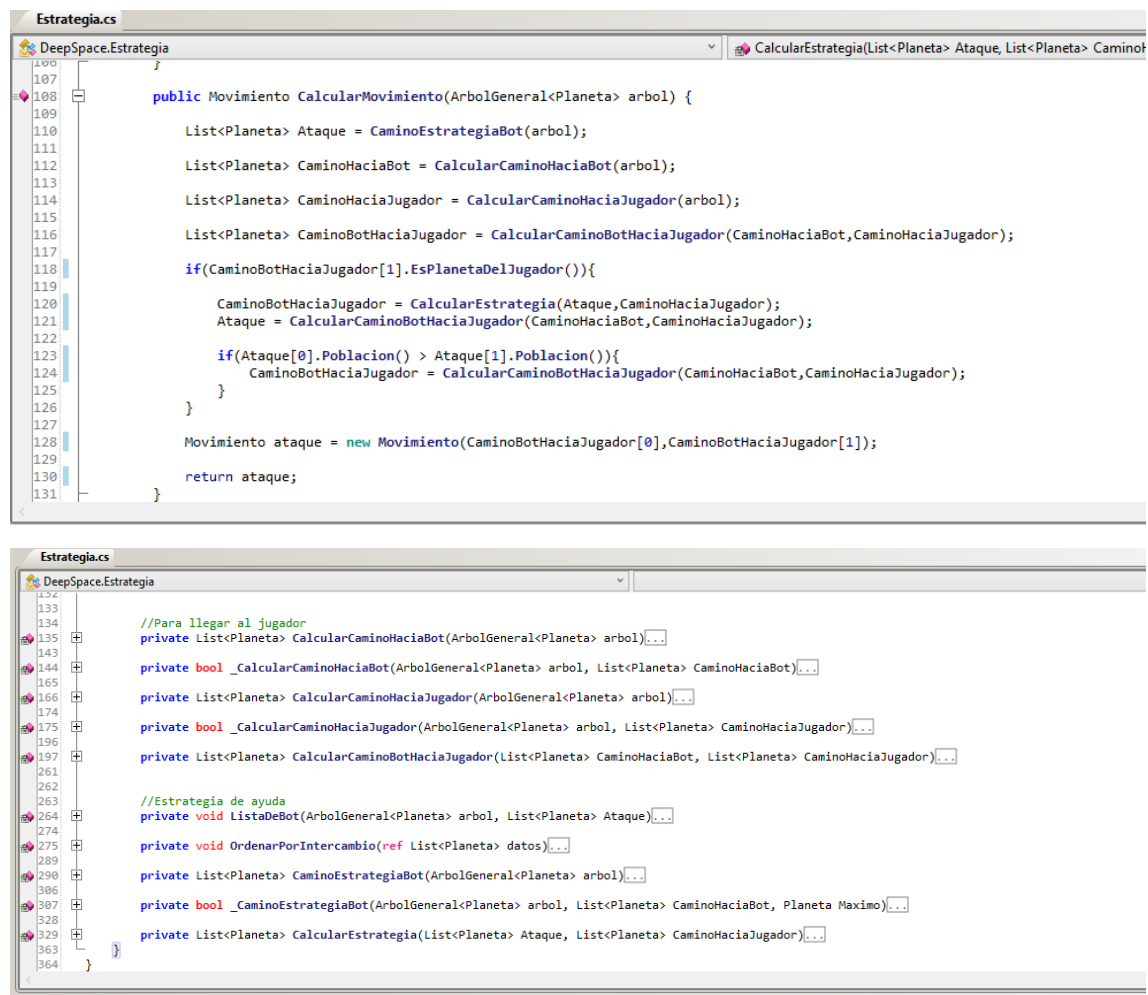
```

Estrategia.cs
DeepSpace.Estrategia
60 public String Consulta3(ArbolGeneral<Planeta> arbol) {
61
62     Cola<ArbolGeneral<Planeta>> c = new Cola<ArbolGeneral<Planeta>>();
63     ArbolGeneral<Planeta> arbolAux;
64
65     int nivel = 0;
66     int poblacionTotal = 0;
67     int contador = 0;
68     int promedio = 0;
69
70     string texto = "Consulta 3: ";
71
72     c.encolar(arbol);
73     c.encolar(null);
74
75     texto = texto + "Nivel: " + nivel;
76     texto = texto + " | PoblacionTotal: " + arbol.getDatosRaiz().Poblacion();
77     texto = texto + " | Promedio: " + arbol.getDatosRaiz().Poblacion();
78
79     while(!c.esVacia()){
80         arbolAux = c.desencolar();
81
82         if(arbolAux == null){
83             if(!c.esVacia()){
84                 nivel++;
85                 texto = texto + "\n          Nivel: " + nivel;
86                 texto = texto + " | PoblacionTotal: " + poblacionTotal;
87                 texto = texto + " | Promedio: " + promedio;
88                 c.encolar(null);
89                 poblacionTotal = 0;
90                 promedio = 0;
91                 contador = 0;
92             }
93         }
94         else{
95             foreach(var hijo in arbolAux.getHijos()){
96                 contador++;
97                 poblacionTotal = poblacionTotal + hijo.getDatosRaiz().Poblacion();
98                 c.encolar(hijo);
99             }
100             if(arbolAux.getHijos().Count > 0){
101                 promedio = poblacionTotal / contador;
102             }
103         }
104     }
105     return texto;

```

Este método comienza similar a los anteriores, con la diferencia que acá lo que buscamos es calcular y retornar población total y promedio por cada nivel. En este caso a la cola donde guardamos el árbol recibido por parámetro encolamos un null que sirve como separador de niveles, donde también tenemos un árbol auxiliar para ir controlando los datos, instanciación de variables en 0 y un texto para ir juntando los datos pedidos. El bucle hasta que no este vacía la cola, empezamos a desencolar en el árbol aux, si el arbol aux no es nulo, en caso de no serlo incrementamos el nivel y empezamos a cargarle información a la variable texto con el nivel, la población y el promedio, encolamos otro null para separar el nivel y declaramos todo en 0 nuevamente para cuando ingrese nuevamente al bucle cargue la información, hasta el caso que sea nulo el árbol aux. Caso que sea nulo se le pedirá a el árbol los hijos que tuviera para ser encolado y luego ser procesados, previo a procesarlos la población va a ser brindada por los métodos de población.

### Calcular movimiento:



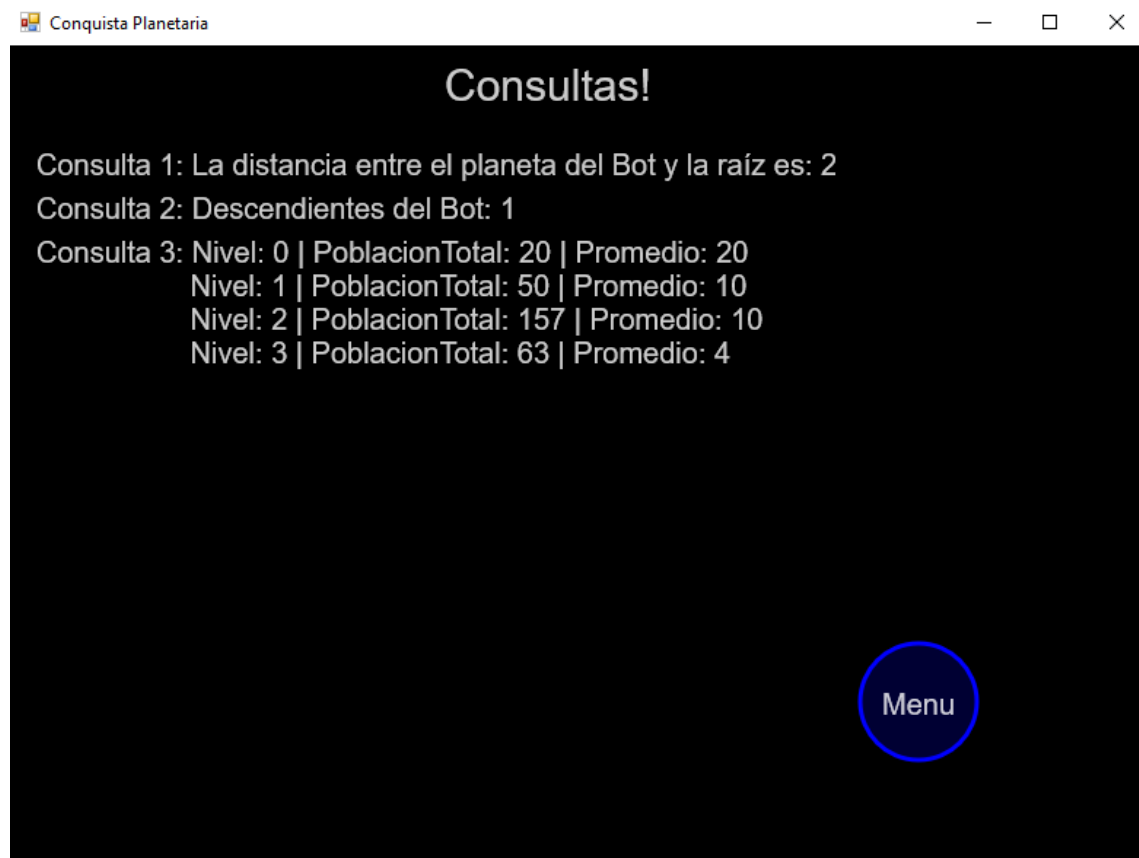
```
107  
108 public Movimiento CalcularMovimiento(ArbolGeneral<Planeta> arbol) {  
109  
110     List<Planeta> Ataque = CaminoEstrategiaBot(arbol);  
111  
112     List<Planeta> CaminoHaciaBot = CalcularCaminoHaciaBot(arbol);  
113  
114     List<Planeta> CaminoHaciaJugador = CalcularCaminoHaciaJugador(arbol);  
115  
116     List<Planeta> CaminoBotHaciaJugador = CalcularCaminoBotHaciaJugador(CaminoHaciaBot, CaminoHaciaJugador);  
117  
118     if(CaminoBotHaciaJugador[1].EsPlanetaDelJugador()){  
119  
120         CaminoBotHaciaJugador = CalcularEstrategia(Ataque, CaminoHaciaJugador);  
121         Ataque = CalcularCaminoBotHaciaJugador(CaminoHaciaBot, CaminoHaciaJugador);  
122  
123         if(Ataque[0].Poblacion() > Ataque[1].Poblacion()){  
124             CaminoBotHaciaJugador = CalcularCaminoBotHaciaJugador(CaminoHaciaBot, CaminoHaciaJugador);  
125         }  
126     }  
127  
128     Movimiento ataque = new Movimiento(CaminoBotHaciaJugador[0], CaminoBotHaciaJugador[1]);  
129  
130     return ataque;  
131 }  
132  
133  
134 //Para llegar al jugador  
135 private List<Planeta> CalcularCaminoHaciaBot(ArbolGeneral<Planeta> arbol) {...}  
136  
137 private bool _CalcularCaminoHaciaBot(ArbolGeneral<Planeta> arbol, List<Planeta> CaminoHaciaBot) {...}  
138  
139 private List<Planeta> CalcularCaminoHaciaJugador(ArbolGeneral<Planeta> arbol) {...}  
140  
141 private bool _CalcularCaminoHaciaJugador(ArbolGeneral<Planeta> arbol, List<Planeta> CaminoHaciaJugador) {...}  
142  
143 private List<Planeta> CalcularCaminoBotHaciaJugador(List<Planeta> CaminoHaciaBot, List<Planeta> CaminoHaciaJugador) {...}  
144  
145 //Estrategia de ayuda  
146 private void ListaDeBot(ArbolGeneral<Planeta> arbol, List<Planeta> Ataque) {...}  
147  
148 private void OrdenarPorIntercambio(ref List<Planeta> datos) {...}  
149  
150 private List<Planeta> CaminoEstrategiaBot(ArbolGeneral<Planeta> arbol) {...}  
151  
152 private bool _CaminoEstrategiaBot(ArbolGeneral<Planeta> arbol, List<Planeta> CaminoHaciaBot, Planeta Maximo) {...}  
153  
154 private List<Planeta> CalcularEstrategia(List<Planeta> Ataque, List<Planeta> CaminoHaciaJugador) {...}  
155 }  
156 }  
157 }  
158 }  
159 }  
160 }  
161 }  
162 }  
163 }  
164 }
```

Creo que acá se ve reflejado la cantidad de problemas que tuve para encarar el juego, me doy cuenta que podría haber sido mucho más simple a la hora de implementar el movimiento, simplemente ahora entiendo que había que hacer bien. Pero quedo así y creo que esta bien...

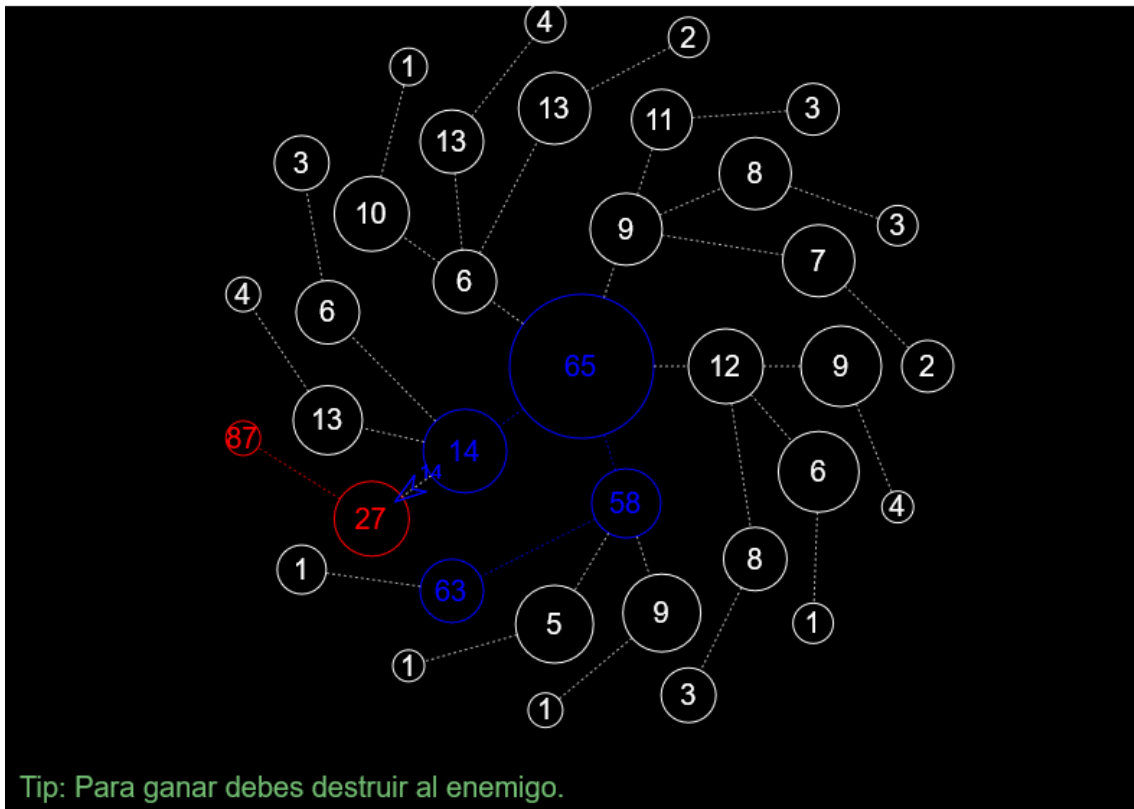
Partimos del método de Calcular movimiento el cual va a tener una secuencia de lista de planetas obtenidas por métodos creados internamente, que nos van a brindar el camino del Bot, del jugador y del Bot al jugador, para así poder implementar el ataque con la condiciones de que si el camino del Bot hacia el jugador es planeta del jugador y así calcular el ataque mediante el ancestro común(algoritmo dentro de calcular estrategia) y luego ese camino enviarlo por parámetro a calcular el camino del Bot al jugador, también para la población del ataque vemos si la raíz es más grande que el hijo recalcula el camino y actualiza la variable del camino.

### **Imágenes y descripciones:**

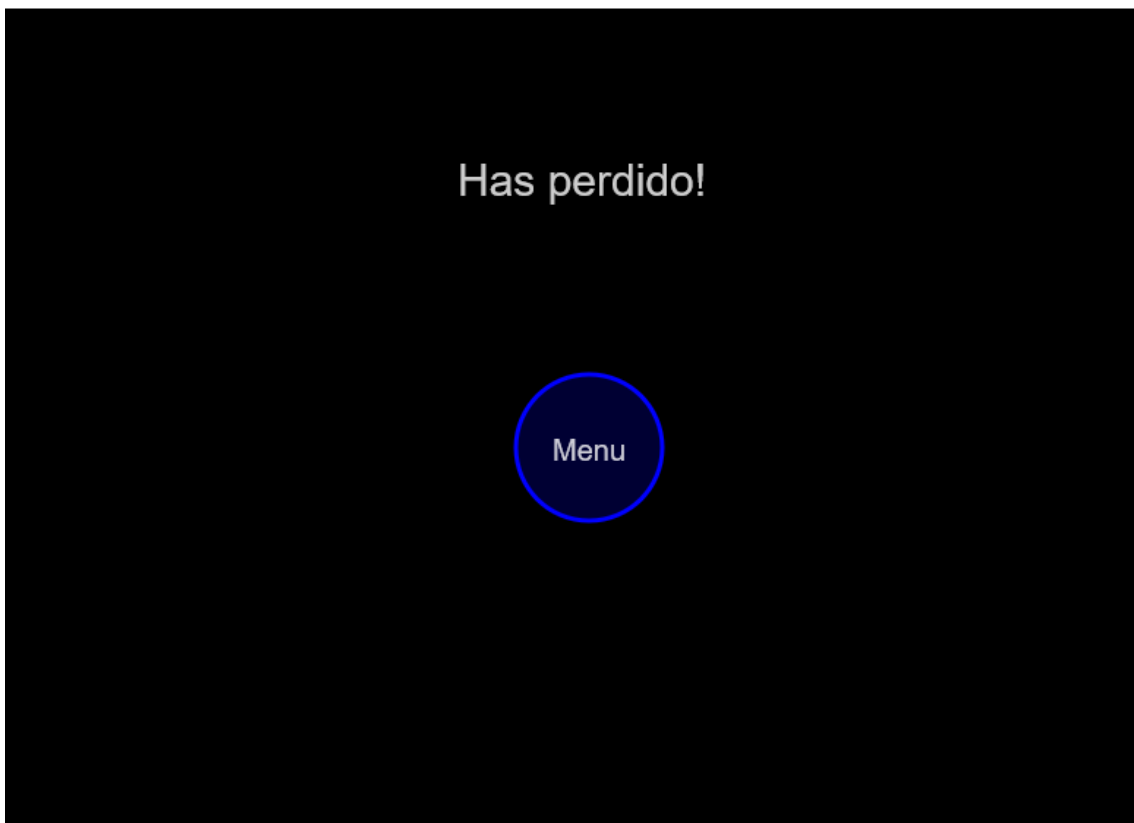
Imágenes de la aplicación en funcionamiento



En la siguiente imagen vemos en funcionamiento la búsqueda del ancestro común



¡HAS PERDIDO!



**Ideas y Sugerencias para mejorar:**

Una idea para mejorar el proyecto seria agregarle funcionalidad para que el movimiento del Bot sea más rápido, ya que vemos que el usuario puede hacer más “clicks” rápidos que el Bot y por eso podría siempre ganar el usuario. Otra mejora podría ser que no siempre busque atacar el ancestro común, sino que compara si le conviene rehacer el camino o seguir expandiéndose con los planetas neutrales y llenarse (con más flota) el planeta de ataque más cercano al ancestro y así atacarlo, porque hay un momento donde el Bot empieza a rehacer el camino y empieza a mandar flotas desde la raíz hasta el planeta más cercano al ancestro.

**Conclusión:**

Estuvo entretenido lograr programar un juego, al principio se hizo muy difícil entender la lógica, pero como antes dicho charlando con compañeros puede ir entendiendo un poco más y empezar a implementar los métodos, gracias a las pruebas y errores, esto ayuda mucho al aprendizaje aparte a mí me sirve como experiencia para aplicar algunos conceptos visto en la materia y aprender a mejorar la lógica.