



Universidad Nacional
ARTURO JAURETCHE



WHO IS WHO - INFORME

Complejidad Temporal, Estructuras de Datos y Algoritmos - Comisión 4
Trabajo Práctico Final - Diego Montaña - UNAJ

Introducción:

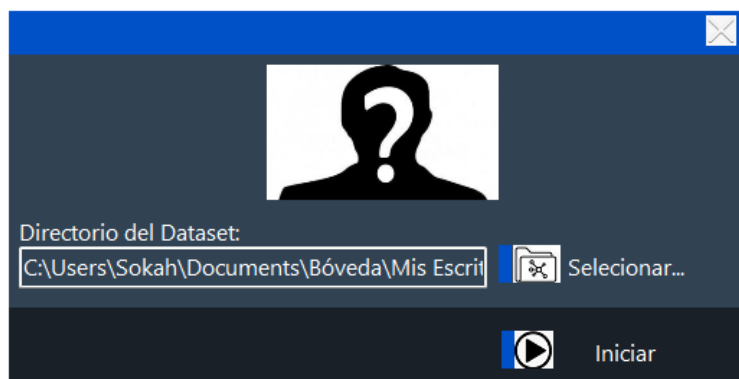
En el presente informe se describe la implementación de una estrategia de Inteligencia Artificial basada en árboles de decisión para el Bot del juego “Who is Who” (WiW). Este Juego se basa en adivinar el personaje que eligió el jugador rival mediante preguntas que pueden describir la identidad del personaje.

La empresa nos “contrató” para implementar la estrategia que utiliza el Bot del juego para encontrar/adivinar el personaje que eligió el usuario. La empresa nos proporcionó un proyecto avanzado, con una interfaz funcional y varias clases implementadas, nuestro trabajo solo se basa en la implementación de la estrategia del bot para que sea funcional, dicha implementación se describe a lo largo del informe.

Proyecto y situación inicial:

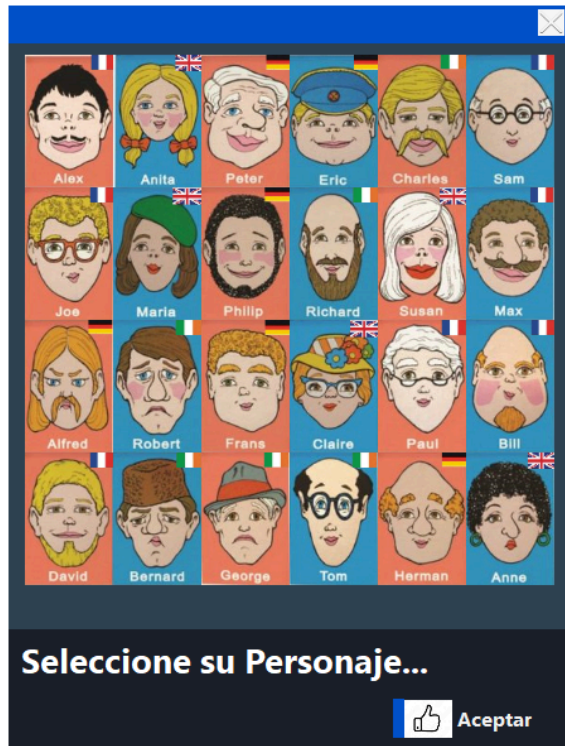
El proyecto fue construido con el entorno de desarrollo SharpDevelop, con Windows Forms para la creación de la interfaz de usuario y con C# como lenguaje de programación principal. El proyecto contaba con el proyecto de juego avanzado, varias clases implementadas, así como la interfaz de usuario funcionando.

El juego cuenta con varias pantallas como las siguientes:



Pantalla inicial:

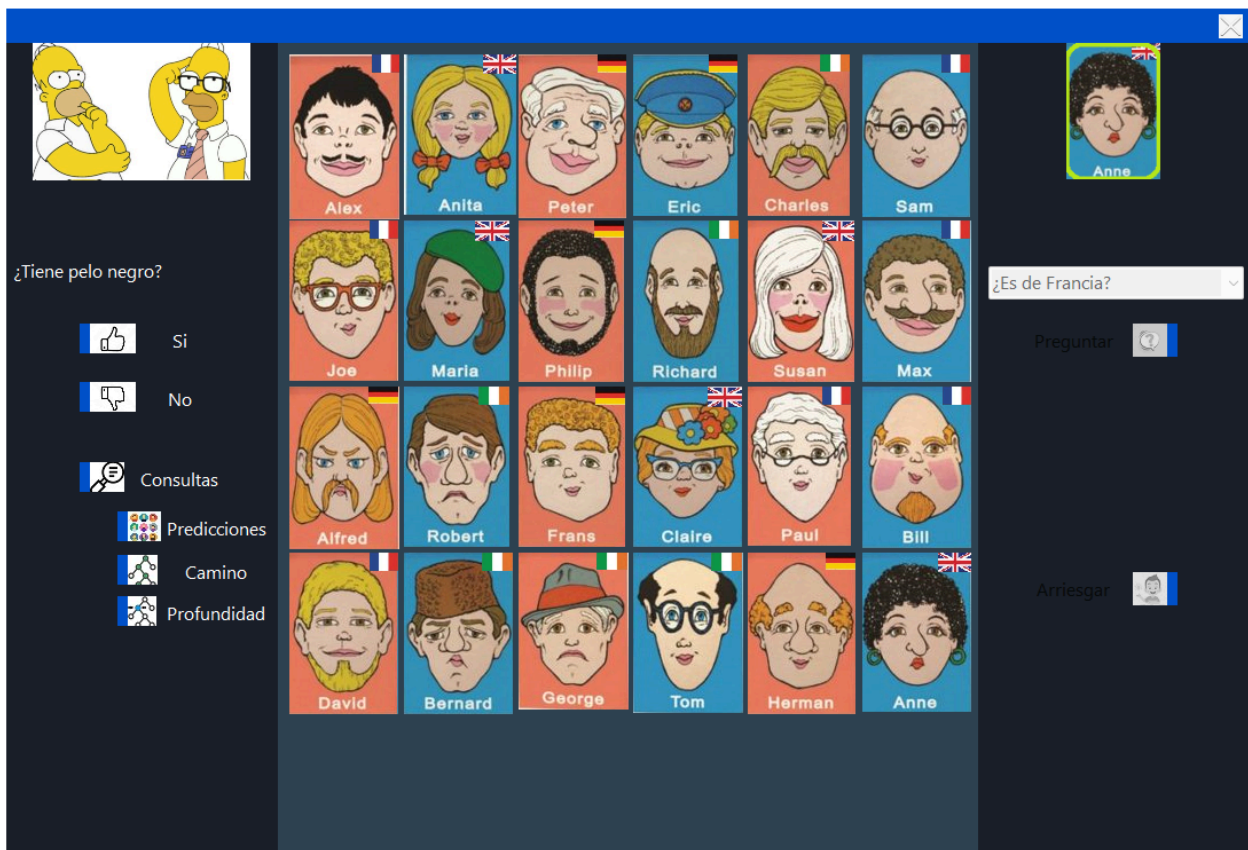
En esta pantalla se pide la ruta del dataset para que se pueda iniciar el juego.



Pantalla de selección de personaje:

En esta pantalla el usuario podrá elegir entre los más de 24 personajes para poder jugar contra el bot del juego.

Pantalla de Juego: La pantalla siguiente después de elegir el personaje, en esta pantalla podemos ver las preguntas que hace el bot para adivinar el personaje que se el usuario eligió. El usuario puede responder a las preguntas haciendo click en los botones del lado izquierdo de la pantalla; Si (pulgar arriba), No (pulgar hacia abajo). Además podrá preguntar al bot por medio de preguntas ya preestablecidas para intentar adivinar la identidad del personaje que eligió el bot. Además el usuario podrá tachar haciendo click en las imágenes de los personajes para ir descartando a los personajes que no se ajustan al personaje del bot según las respuestas que él mismo.



En esta pantalla podemos ver un apartado que tienen que ver con el trabajo que debemos hacer, este apartado es "Consultas", en este apartado se desprenden 3 botones, abren 3 ventanas en el cual podremos mostrar de forma visual el resultado de las consultas a la estrategia.

Clases:

Las clases disponibles inicialmente fueron; ArbolBinario, Backend, Clasificador, Cola, ConjuntoDeDatos, DesicionData, Estrategia (a implementar) ,Particionador , Pregunta y además de otras clases igual de importantes que forman parte de la creación interfaz de usuario, a partir de ahora solo nos vamos a centrar en las mencionadas.

ArbolBinario: Clase de árbol binario que almacena un dato tipo genérico T , con un hijo de izquierdo y un hijo derecho de la misma clase ArbolBinario. Se usa en el proyecto como estructura de dato principal para representar el árbol de decisión para el juego who is who.

ArbolBinario
- dato - hijoIzquierdo - hijoDerecho
+ ArbolBinario(dato) + getDatoRaiz() + getHijoIzquierdo() + getHijoDerecho() + agregarHijoIzquierdo(ArbolBinario) + agregarHijoDerecho(ArbolBinario) + eliminarHijoIzquierdo() + eliminarHijoDerecho() + esHoja() + preorden() + inorden() + postorden() + recorridoPorNiveles() + recorridoEntreNiveles(n, m) + contarHojas()

Cola: Clase cola FIFO genérico, los elementos se almacenan en una List<T> datos. y proporciona los métodos necesarios para su uso posterior normal.

Cola
- datos
+ encolar(elem) + desencolar() + tope() + esVacia() + cantidadElementos()

ConjuntoDeDatos: Esta clase carga y almacena los datos de un archivo .csv y los clasifica en los 2 atributos; filas y encabezados, el objetivo de la clase es principalmente este.

ConjuntoDeDatos
-filas -encabezados
- filas - encabezados + ConjuntoDeDatos(filas, encabezados) + ConjuntoDeDatos(pathCsvFile, delimitador) + ConjuntoDeDatos() + obtenerEtiquetaFila(index) + obtenerFila(index) + cantidadFilas() + Filas { get; set; } + Encabezados { get; set; }

Clasificador: Esta clase clasifica a partir de un ConjuntoDeDatos, la pregunta, la predicciones que estarán en las hojas y nos abstrae de la lógica, para solo hacer uso de los métodos y obtener nuevos clasificadores que pertenezcan a la rama izquierda o a la rama derecha.

Clasificador
- resultados - preguntas - ganancia - pregunta - filas - encabezados - datosDeHijos
+ Clasificador(ConjuntoDeDatos) + crearHoja() + obtenerDatoHoja() + obtenerPregunta() + obtenerDatosIzquierdo() + obtenerDatosDerecho() + obtenerClasificadorIzquierdo() + obtenerClasificadorDerecho()

Particionador: Envuelve la lógica que ayuda a la construcción de los clasificadores.

Particionador
+ class_counts(rows) + partition(rows, question) + find_best_split(rows, header) - gini(rows) - info_gain(left, right, current_uncertainty)

DecisionData: Esta clase contiene la información que se almacenará en cada nodo para construir el árbol de decisión, en sí almacena la Pregunta y las predicciones que se almacena en los nodos hojas. siendo las predicciones la respuesta final a la question.

DecisionData
- question - predictions
+ DecisionData(Pregunta) + DecisionData() + Question { get; } + Predictions { get; } + ToString() - imprimirPredicciones()

Pregunta: Esta clase almacena la condición (pregunta) y permite comparar la coincidencia de las respuestas, además de proporcionar la pregunta en formato texto para mostrar a los usuarios.

Pregunta
- columna - valor - encabezado - texto
+ Pregunta(columna,valor,encabezado) + coincide(ejemplo) + coincide(dataset,fila) + coincide(val) + ToString() + TextoParaUsuario() + Texto { get; set; } + { get; set; }

Backend: Esta clase inicializa toda la lógica en el juego, carga el archivo .csv en un ConjuntoDeDatos, hace uso de la Clase Estrategia , para crear el árbol de decisiones de tipo ArbolBinario<DecisionData>, y hacer las consultas que faltan por implementar en Estrategia, además obtiene las respuestas del usuario y las procesa para seguir avanzando en el juego .

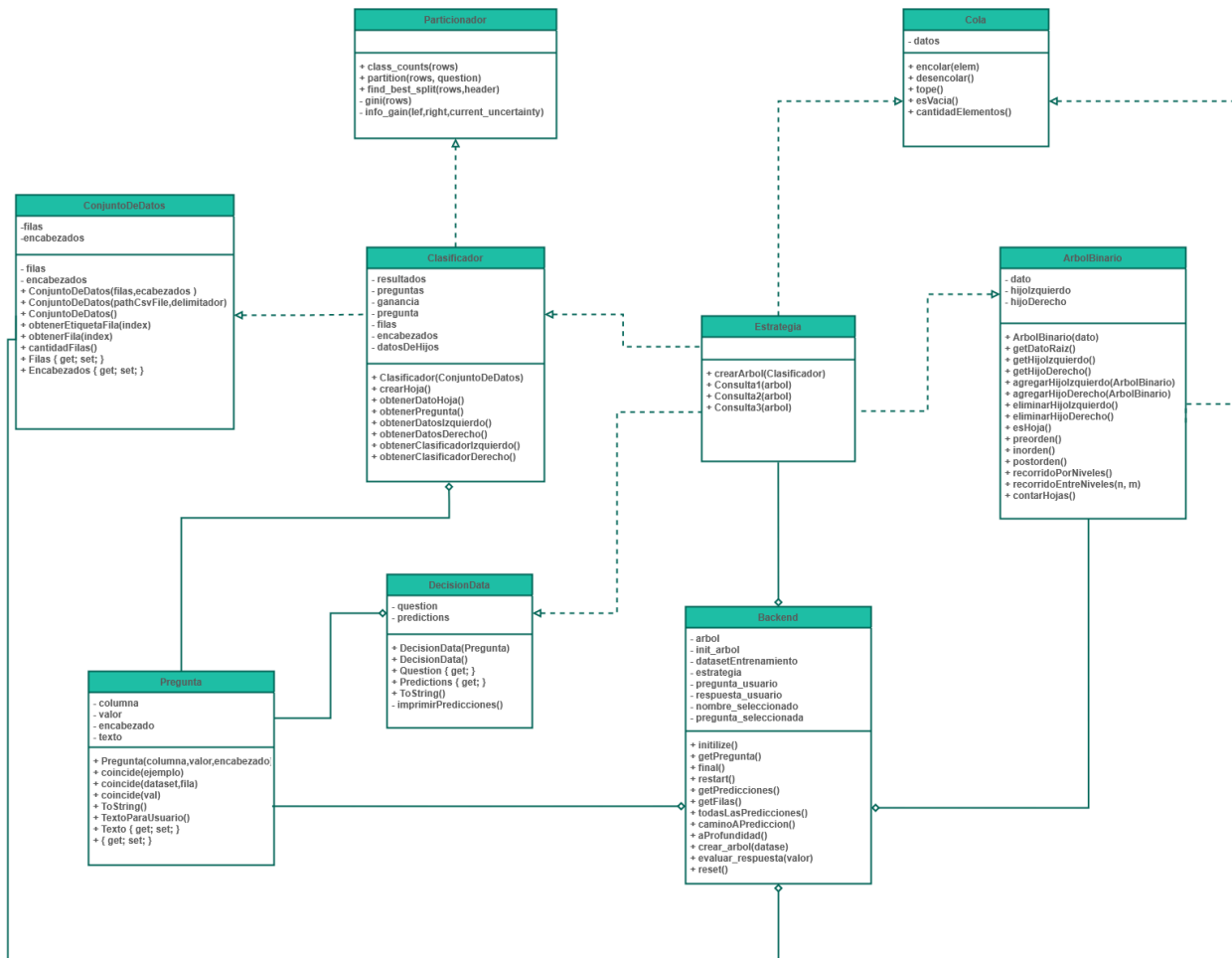
Backend
<ul style="list-style-type: none"> - arbol - init_arbol - datasetEntrenamiento - estrategia - pregunta_usuario - respuesta_usuario - nombre_seleccionado - pregunta_seleccionada
<ul style="list-style-type: none"> + initilize() + getPregunta() + final() + restart() + getPredicciones() + getFilas() + todasLasPredicciones() + caminoAPrediccion() + aProfundidad() + crear_arbol(datasase) + evaluar_respuesta(valor) + reset()

Estrategia: En esta clase se prevee que este la lógica para poder crear el árbol decisión, con el uso de las clases previamente descritas, además dispone de 3 métodos de consulta que deben implementarse.

Estrategia
<ul style="list-style-type: none"> + crearArbol(Clasificador) + Consulta1(arbol) + Consulta2(arbol) + Consulta3(arbol)

Diagrama de Clases:

Con las clases dadas se planteó el siguiente diagrama de clases:



Implementación de la Estrategia:

A partir del estudio del proyecto y el entendimiento de las clases ya implementadas se procedió a implementar los métodos en la clase estrategia según los requerimientos de la empresa.

Método CrearArbol: En primer lugar la empresa planteó que este método debería crear una instancia de un árbol de decisión a partir de un clasificador que se envía por parámetro y finalmente retorna una instancia de `ArbolBinario<DecisionData>`.

Con el objetivo claro del método se le planteó el siguiente código que cumple con la petición de la empresa:

```
1  /* -----
2  1. CrearArbol (Clasificador clasificador): Este método construye un árbol de decisión a
3  partir de un clasificador que es enviado como parámetro y retorna una instancia de
4  ArbolBinario<DecisionData>.
5  */
6  public ArbolBinario<DecisionData> CrearArbol(Clasificador clasificador)
7  {
8      // si es clasificador es hoja
9      if (clasificador.crearHoja()){
10         // predicciones en nodo hoja
11         Dictionary<string, int> datosHoja = clasificador.obtenerDatoHoja();
12         return new ArbolBinario<DecisionData>(new DecisionData(datosHoja));
13     }else{
14         // se guarda la pregunta en una variable, con ella se crea una DecisionData y un nuevo arbol
15         Pregunta preguntaActual = clasificador.obtenerPregunta();
16         ArbolBinario<DecisionData> arbol = new ArbolBinario<DecisionData>(new DecisionData(preguntaActual));
17         // se obtiene los clasificadores y se hace llamadas recursivas
18         // para crear un arbol hijo izquierdo y un arbol hijo derecho con ellos
19         Clasificador clasificadorIzq = clasificador.obtenerClasificadorIzquierdo();
20         Clasificador clasificadorDer = clasificador.obtenerClasificadorDerecho();
21         arbol.agregarHijoIzquierdo(CrearArbol(clasificadorIzq));
22         arbol.agregarHijoDerecho(CrearArbol(clasificadorDer));
23         return arbol;
24     }
25 }
```

Dado la naturaleza de los arboles binarios, se optó por hacer el método recursivo, para construir el árbol de decisión completo nodo a nodo a partir del clasificador, por lo tanto se plantea el siguiente caso base; si es el clasificador es candidato a ser hoja, se obtiene los datos hoja del clasificador y con él se crea un árbol binario de tipo `DecisionData` y se retorna un `ArbolBinario<DecisionData>`.

En el caso recursivo (dentro del bloque de código de else, línea 14 a 23), si el clasificador no es candidato a ser hoja, entonces es una pregunta, por lo tanto se crea un variable de tipo Pregunta y se guarda en ella la pregunta que contiene el clasificador, con la pregunta se crea un nuevo árbol de decisión y finalmente se obtiene los clasificadores izquierdo y derecho del clasificador pasado por parámetro, y se finaliza haciendo dos llamadas recursivas para crear un árbol izquierdo y derecho que se agregaran como hijos del árbol previamente creado en los pasos anteriores (línea 16).

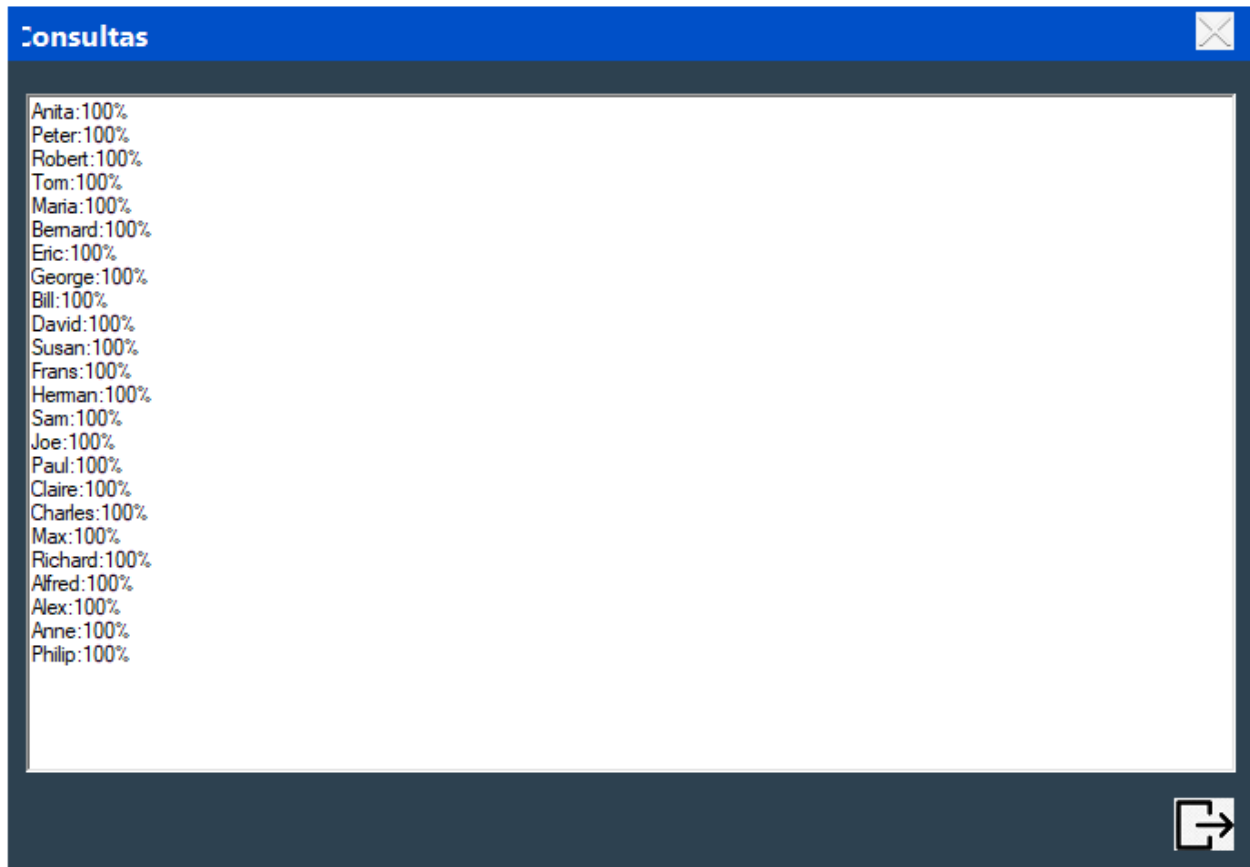
Método Consulta1: La empresa nos planteó que este método debería retornar todas las predicciones que contiene el árbol de decisiones pasado por parámetro, dado este nuevo objetivo se construyó el siguiente código:

```
1  /* -----
2  2. Consultar(ArbolBinario< DecisionData > arbol): Retorna un texto con todas las posibles
3  predicciones que puede calcular el árbol de decisión del sistema
4  */
5  public String Consultar(ArbolBinario<DecisionData> arbol)
6  {
7      // se hace uso de un metodo que recorre el arbol en preorden modificado,
8      // al metodo se le pasa por parametro el arbol a recorrer y un string pasado por ref donde
9      // quiero que se concatene todas las predicciones que estan en los nodos hojas.
10     string resultado="";
11     ObtenerPredicciones(arbol, ref resultado);
12     return resultado;
13 }
14 private void ObtenerPredicciones(ArbolBinario<DecisionData> arbol, ref string predicciones){
15     if(arbol == null){
16         return;
17     }
18     if (!arbol.esHoja()){
19         if (arbol.getHijoIzquierdo() != null){
20             ObtenerPredicciones(arbol.getHijoIzquierdo(), ref predicciones);
21         }
22         if(arbol.getHijoDerecho() != null){
23             ObtenerPredicciones(arbol.getHijoDerecho(), ref predicciones);
24         }
25     }else{
26         // se concatena las predicciones de un nodo hojas, en un string "predicciones" pasado por ref
27         // se obtiene el dato raiz de una hoja que debe de ser un DecisionData
28         // y se obtienen las predicciones con el metodo ToString();
29         predicciones += arbol.getDatoRaiz().ToString() + "\n";
30     }
31 }
```

Dado que las predicciones se encuentran en los nodos hojas, en objetivo del código no es mas que recorrer el árbol hasta todas sus hojas, por lo tanto se plantea un nuevo método

privado que recorre el árbol en su totalidad y que en el fondo es un recorrido en preorden con ciertas modificaciones, en primer lugar se plantea que el método tenga 2 parámetros el árbol de decisiones dado y un string pasado por referencia llamado "predicciones" en cual el objetivo es concatenar las predicciones de los nodos hojas en el. Finalmente se obtiene el resultado en el método consulta 1 y es retornado.

En la siguiente imagen se puede ver el resultado retornado por la consulta1:



Método Consulta2: En este método la empresa nos pide que se retorne un texto que contenga todos los caminos posibles en un árbol de decisiones, hasta cada predicción (nodo hoja). Dado este planteo se construyó el siguiente código:

```
1  /* -----
2  3. Consulta2(ArbolBinario< DecisionData > arbol): Retorna un texto que contiene todos los caminos
3  hasta cada predicción.
4  */
5  public String Consulta2(ArbolBinario<DecisionData> arbol)
6  {
7      // se obtiene los caminos en una lista y luego se concatena los caminos a un string para ser mostrado
8      // ademas como plus se "dibuja" el arbol de dicision para que se vea el resultado de forma mas visual.
9      string camino = "";
10     List<string>caminos= new List<string>();
11     string todosLosCaminosHastaPrediccion = "";
12     string r= DibujarArbol(arbol, "", "");
13
14     ObtenerCaminos(arbol,ref camino,ref caminos);
15     foreach ( string c in caminos){
16         todosLosCaminosHastaPrediccion += c ;
17     }
18     return todosLosCaminosHastaPrediccion +"\n"+ r;
19 }
20
21
22 private void ObtenerCaminos(ArbolBinario<DecisionData> arbol, ref string camino,ref List<string> caminos){
23     // una variacion del metodo obtenerPredicciones() , este ademas de recorrer hasta las hojas(predicciones),
24     // concatena el nodo(Pregunta) en el camino hasta llegar a una hoja, obteniendo asi todo el camino hasta una hoja
25     // se agregan cada camino a la lista de caminos.
26     if(arbol == null){
27         return;
28     }
29     if (!arbol.esHoja()){
30         string caminoAnterior = camino;
31         string pregunta = arbol.getDatosRaiz().Question.Texto; //.TextoParaUsuario();
32         if (arbol.getHijoIzquierdo() != null){
33             camino = camino + pregunta + "= NO -> ";
34             ObtenerCaminos(arbol.getHijoIzquierdo(),ref camino, ref caminos);
35         }
36         camino = caminoAnterior;
37         if(arbol.getHijoDerecho() != null){
38             camino = camino + pregunta + "= SI -> ";
39             ObtenerCaminos(arbol.getHijoDerecho(),ref camino,ref caminos);
40         }
41         camino = caminoAnterior;
42     }else{
43         string prediccion = arbol.getDatosRaiz().ToString();
44         caminos.Add(camino + prediccion + "\n");
45     }
46 }
```

Para resolver este planteo, se optó por hacer un método nuevo muy parecido al que se usó en la resolución de la consulta1, que además de hacer un recorrido hasta todas las hojas, va construyendo un camino que va desde el nodo raíz hasta un nodo hoja y cuando se llega al nodo hoja el camino construido se agrega a una lista de caminos.

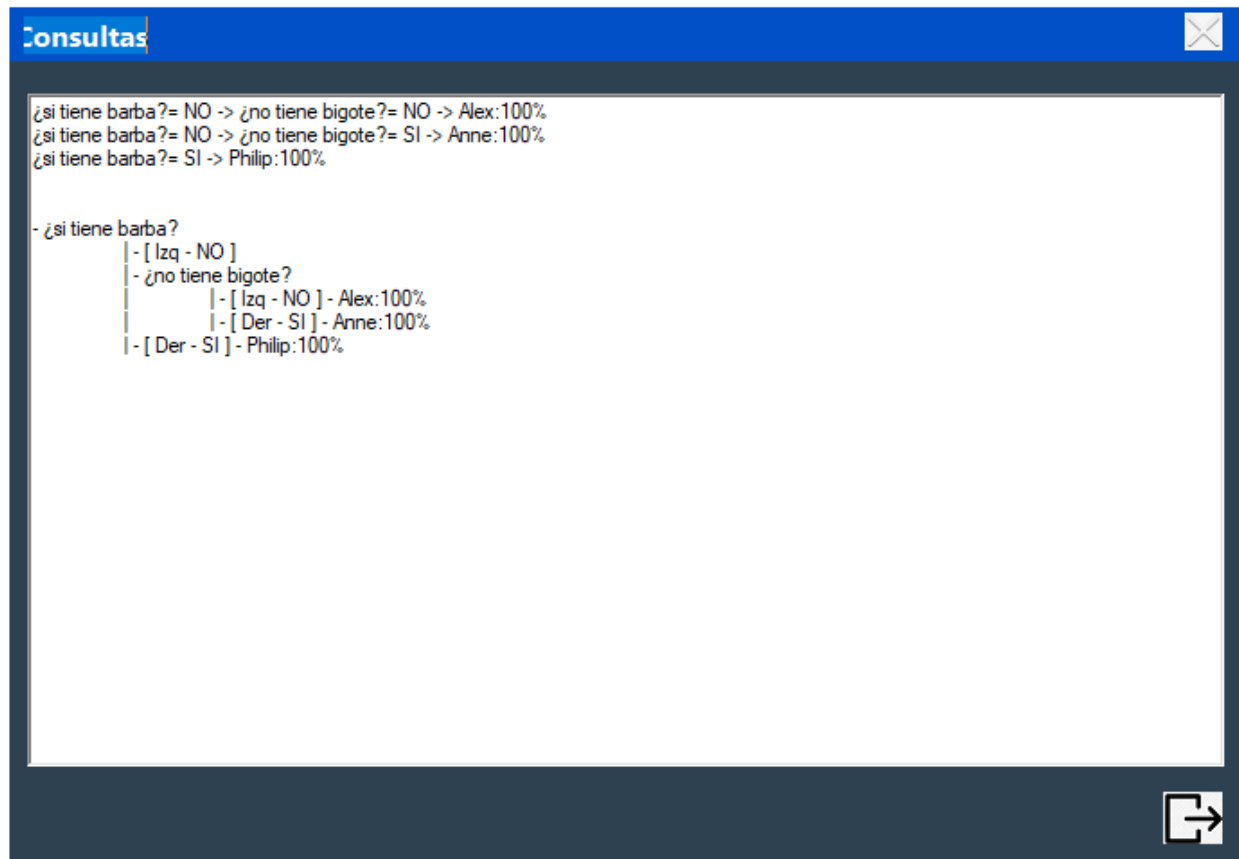
El nuevo método privado obtiene como parámetros un `ArbolBinario<DecisionData> arbol`, `string camino` pasado por referencia y una `list<String> caminos` igualmente pasado por referencia. A diferencia del método anterior, aquí se debe diferenciar claramente el recorrido que se hace, por lo que esto se hace, obteniendo el texto de la pregunta del nodo actual y concatenarlo en el camino y además concatenando un mensaje que indica si era hijo izquierdo ("No ->) o derecho ("Si -> ") siendo estas respuestas a la pregunta del nodo actual. Finalmente en el caso base, se concatena el camino con la predicción del nodo hoja y se añade a la lista de caminos. Nuevamente en el método `consulta2`, se recorre la lista con todos los caminos y se concatena en una nueva variable.

Como plus y siguiendo en el mismo planteo se consideró crear un nuevo método que tiene como objetivo "dibujar" el árbol de decisión por completo, este método sigue la misma lógica del recorrido, pero además añade indentaciones y caracteres para su fácil visualización como un árbol binario. Finalmente el resultado de este método es concatenado junto al resultado anterior y es retornado.

En la siguiente imagen puede verse el código construido para "dibujar" el árbol de decisiones completo:

```
1 private string DibujarArbol( ArbolBinario<DecisionData> arbol, string indent, string rama){
2     if (arbol == null){
3         return "";
4     }
5     string resultado = indent ;
6     if (rama == "Izq - NO" ){
7         resultado += "- [ "+rama+" ]" ;
8     }else if ( rama == "Der - SI"){
9         resultado += "| - [ "+rama+" ]" ;
10    }
11    if (!arbol.esHoja()){
12        resultado += "\n" + indent + "- "+arbol.getDatoRaiz().Question.Texto +"\n";
13    }else{
14        resultado += " - "+ arbol.getDatoRaiz().ToString() +"\n";
15    }
16
17    if (arbol.getHijoIzquierdo() != null){
18        string nuevoIndent = indent + (arbol.getHijoDerecho() != null ? " |":" " );
19        resultado += DibujarArbol(arbol.getHijoIzquierdo(), nuevoIndent, "Izq - NO") ;
20    }
21    if (arbol.getHijoDerecho() != null){
22        string nuevoIndent = indent + " " ;
23        resultado += DibujarArbol(arbol.getHijoDerecho(), nuevoIndent, "Der - SI") ;
24    }
25    return resultado;
26 }
```

En la siguiente imagen se puede ver el texto con todos los caminos (preguntas y predicciones) además en la parte inferior puede verse el arbol “dibujado”:



Aclaración: se respondió una pregunta para que el resultado de la consulta2 pueda verse por completo en el screenshot actual.

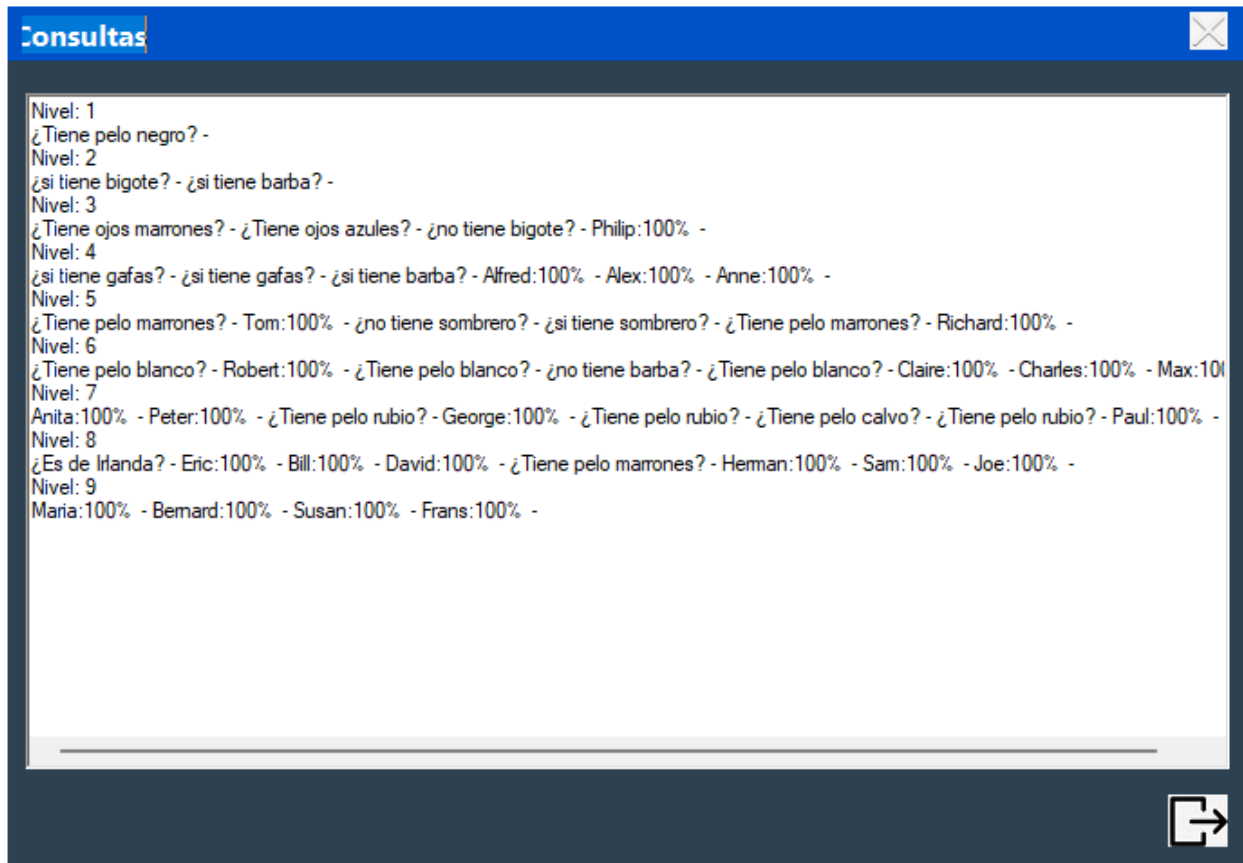
Método Consulta3: En este caso la empresa requirió que este método retorne un texto que contenga los datos de los nodos (preguntas y predicciones) pero diferenciando en que nivel del árbol se encuentran.

Para este planteamiento no se optó por crear nuevos métodos y se optó por hacer uso de las lógicas de recorrido entre niveles de los árboles binarios para cumplir con lo requerido, solo bastó con hacer ciertas modificaciones y obtener los datos de cada nodo sólo

diferenciándolos con los nodos hojas para obtener las predicciones de los nodos no hojas que contiene las preguntas y concatenarlos a la variable que queremos que se retorne. Además se necesitó de una cola para poder saber en qué nivel del árbol nos encontramos. En la siguiente imagen se puede ver el código que se construyó para resolver este caso:


```
1  /* -----
2  4. Consulta3 (ArbolBinario< DecisionData > arbol): Retorna un texto que contiene los datos
3  almacenados en los nodos del árbol diferenciados por el nivel en que se encuentran.
4  */
5  public String Consulta3(ArbolBinario<DecisionData> arbol)
6  {
7      // se usa la logica de recorrido entre niveles para mostrar el resultado que se quiere.
8      string resultado;
9      Cola<ArbolBinario<DecisionData>> colaArboles = new Cola<ArbolBinario<DecisionData>>();
10     Cola<int> colaNiveles = new Cola<int>();
11     int nivelActual = 1;
12     colaArboles.encolar(arbol);
13     colaNiveles.encolar(nivelActual);
14     resultado = "Nivel: " + nivelActual + "\n";
15     while(!colaArboles.esVacia()){
16         ArbolBinario<DecisionData> arbolActual = colaArboles.desencolar();
17         int nivelArbol = colaNiveles.desencolar();
18         if (nivelArbol > nivelActual){
19             nivelActual = nivelArbol;
20             resultado += "\n" + "Nivel: " + nivelActual + "\n" ;
21         }
22         if (!arbolActual.esHoja()){
23             resultado += arbolActual.getDatosRaiz().Question.Texto + " - ";
24         }else{
25             resultado += arbolActual.getDatosRaiz().ToString() + " - ";
26         }
27         if( arbolActual.getHijoIzquierdo() != null){
28             colaArboles.encolar(arbolActual.getHijoIzquierdo());
29             colaNiveles.encolar(nivelActual + 1);
30         }
31         if (arbolActual.getHijoDerecho() != null){
32             colaArboles.encolar(arbolActual.getHijoDerecho());
33             colaNiveles.encolar(nivelActual + 1);
34         }
35     }
36     return resultado;
37 }
```


En la siguiente imagen se puede ver el resultado de la consulta3, primero se muestra el nivel actual y siguiente los nodos de ese nivel separando con guiones cada nodo (pregunta o predicción) :



Problema Encontrado:

Durante la implementación de los métodos se encontró que los nodos izquierdos no corresponden a las respuestas falsas de la pregunta anterior o del nodo padre. por lo que se optó por hacer un pequeño cambio en el método partition de la clase Particionador, como se puede ver en la siguiente imagen:



```
1 public static IList<IList<IList<string>>> partition(IList<IList<string>> rows, Pregunta question){
2     //Partitions a dataset.
3     // For each row in the dataset, check if it matches the question. If
4     // so, add it to 'true rows', otherwise, add it to 'false rows'.
5
6     List<IList<string>> ltrue_rows = new List<IList<string>>();
7     List<IList<string>> lfalse_rows = new List<IList<string>>();
8
9     foreach (var row in rows) {
10         if (question.coincide(row))
11             ltrue_rows.Add(row);
12         else
13             lfalse_rows.Add(row);
14     }
15
16     IList<IList<IList<string>>> res = (IList<IList<IList<string>>>) new List<IList<IList<string>>>();
17     //res.Add(ltrue_rows);
18     //res.Add(lfalse_rows);
19
20     // ** se modifiko esto para que los rows false esten a la izquierda,
21     // para que a la hora de crear el arbol los nodos que esten como hijos izquierdos
22     // correspondan a las "valores" falsos
23     res.Add(lfalse_rows);
24     res.Add(ltrue_rows);
25
26     return res;
27 }
```

Este pequeño cambio bastó para solucionar este problema.

Conclusión: Finalizado el trabajo pude entender y reforzar los conocimientos sobre árboles binarios y aprender sobre los árboles de decisiones y como su funcionamiento ayuda en el entrenamiento de bots como el de este juego.