TRABAJO FINAL

CTEDYA

MIRANDA IBARRA MELINA

Introducción

El presente informe tiene como objetivo integrar los contenidos vistos en la materia. Se expondrá la resolución del proyecto machine learning planteado por la catedra con el fin de desarrollar el procedimiento, exponer los componentes del programa y explicar el funcionamiento del mismo, a demás de integrar los conceptos adquiridos sobre búsqueda de caminos, recorrido de arboles binarios y hacer el uso del TAD cola.

Desarrollo

Para dar comienzo al desarrollo del proyecto se comenzó con la visualización de todas las clases que lo comprenden y se realizó un análisis de lo que se debía hacer. Durante este análisis se pudo observar que es un sistema de toma de decisiones basado principalmente en un árbol binario, el cual se deberá recorrer para hallar sus valores y así, complementar las otras funciones del proyecto para que este sea funcional.

Una vez realizado esto, se comenzó a codificar las funciones requeridas. En el proceso se encontraron los siguientes problemas:

En el comienzo del desarrollo de la función **CrearArbol**

- No saber cómo meterle datos al árbol.
- No saber crear las preguntas para meterlas en el árbol.
- No poder crear el Árbol binario.
- No insertar las hojas al árbol.

Pero luego, gracias a las consultas, al análisis parte por parte de cada elemento del enunciado y gracias a los apuntes tomados en clase se logró realizarlo.

También surgieron los siguientes en la creación de las funciones **Consulta**

- No lograr que la **Consulta1** retorne los resultados de las hojas del árbol.
- Cuando **Consulta1** está a punto de predecir el personaje sale un mensaje de error y se detiene el programa.

Estos fueron resueltos cuando se realizó la inserción de hojas al árbol.

- En la **Consulta2** no saber cómo traer todos los caminos del árbol.

Este problema fue resuelto chequeando si el dato procesado pertenece a una hoja, si lo era, había que agregar la lista camino a la lista caminos, si no, seguir con el proceso preorden.

- En la **Consulta3** que no traiga los datos del árbol separado por niveles.

En esta consulta se estaba retornando 2 veces la variable string “cola” y, además, no se estaba concatenando el texto, solo se estaba reemplazando al anterior por el nuevo. Fue solucionado usando “cola += “.

Problema general

No poder imprimir los textos uno debajo del otro.

Este fue solucionado concatenando un *"\n"* en la variable que almacena los datos.

Pantallas que componen el sistema codificado

Función **CrearArbol**

```
public ArbolBinario<DecisionData>CrearArbol(Clasificador clasificador)
{
    DecisionData pregunta = new DecisionData(clasificador.obtenerPregunta());
    ArbolBinario<DecisionData> arb = new ArbolBinario<DecisionData>(pregunta);
    DecisionData nodohoja;

    if(clasificador.crearHoja()==true)
    {
        nodohoja = new DecisionData(clasificador.obtenerDatoHoja());
        arb = new ArbolBinario<DecisionData>(nodohoja);
    }
    else
    {
        pregunta = new DecisionData(clasificador.obtenerPregunta());
        arb = new ArbolBinario<DecisionData>(pregunta);
        arb.agregarHijoDerecho(CrearArbol(clasificador.obtenerClasificadorDerecho()));
        arb.agregarHijoIzquierdo(CrearArbol(clasificador.obtenerClasificadorIzquierdo()));

    }

    return arb;
}
```

En esta función perteneciente a la clase Estrategia se comenzó creando los objetos que compondrán el árbol binario de decisiones, como lo es *pregunta* (el cual almacena una pregunta brindada por el clasificador), *arb* (que almacena a *pregunta* en su contenido), y *nodohoja* (que almacena el dato de la hoja).

Luego, se pregunta si el conjunto de datos corresponde a un nodo-hoja, cuando es verdadero se almacena en el objeto *nodohoja* el dato de la hoja y este se almacena en *arb*. Cuando es falso, vuelve a almacenar una nueva pregunta en el objeto *pregunta* y la guarda nuevamente en *arb*.

Posteriormente, se agregan los hijos derechos e izquierdos a *arb* haciendo una llamada recursiva con sus respectivos clasificadores como parámetro.

Para finalizar, la función retorna el árbol binario *arb*.

Función **Consulta1**

```
public String Consulta1(ArbolBinario<DecisionData> arbol)
{
    return arbol.contenidoHoja();
}
```

Esta función también perteneciente a la clase Estrategia retorna una función contenidoHoja perteneciente a la clase ArbolBinario.

```
public string contenidoHoja()
{
    string contenido = "";

    if(esHoja())
        contenido = dato.ToString() + " \n ";

    if(hijoIzquierdo!=null)
        contenido =  contenido + hijoIzquierdo.contenidoHoja();

    if(hijoDerecho!=null)
        contenido =   contenido + hijoDerecho.contenidoHoja();


    return contenido;
}
```

Esta retorna una variable "*contenido*" donde se almacena el contenido de la raíz del árbol y luego sus hijos, esto lo hace con un recorrido preorden donde primero procesa la raíz, luego el hijo izquierdo y por último el hijo derecho, los cuales llaman recursivamente a la función.

Función **Consulta2**

```
public String Consulta2(ArbolBinario<DecisionData> arbol)
{
    string todosLosCaminos= "";
    Caminos(arbol,ref camino,ref caminos, ref copia);
    foreach(ArbolBinario<DecisionData> j in caminos)
    {
        if(j.esHoja())
            todosLosCaminos+= "|" + j.getDatoRaiz().ToString() + "|" + "\n";
        else
            todosLosCaminos += "|" + j.getDatoRaiz().ToString();
    }
    return todosLosCaminos;
}
```

Para comenzar, en Consulta2 se utilizó de referencia el método DFS empleado en un ejercicio de grafos que se vio en la cursada y el método preorden, modificando ambos.

Se definió una variable del tipo string para almacenar los datos retornados del recorrido de la lista caminos.

Se llamó a la función Caminos pasando las listas por referencia, así permitiendo que estas modifiquen su valor original.

Luego, se recorrió la lista caminos y se preguntó si el elemento j pertenece a una hoja, si es así almacena en la variable todosLosCaminos el valor de la raíz del elemento, 2 separadores y un salto de línea.

Una vez finalizado el recorrido de todos los elementos de la lista caminos se retorna la variable todosLosCaminos.

La función Caminos quedó codificada de la siguiente forma:

private List<ArbolBinario<DecisionData>> **Caminos**(ArbolBinario<DecisionData> arbol, **ref** List<ArbolBinario<DecisionData>> camino, **ref** List<ArbolBinario<DecisionData>> caminos, **ref** List<ArbolBinario<DecisionData>> copia)

```
{
    if(arbol.getDatoRaiz() !=null)
    {
        camino.Add(arbol);
        if(arbol.esHoja())
        {
            //guarda camino en una lista de copia
            copia.AddRange(camino);
            //copia camino en caminos
            foreach(ArbolBinario<DecisionData> i in camino)
                caminos.Add(i);
             camino.RemoveAt(camino.Count - 1);
             return camino;
        }
        if(arbol.getHijoIzquierdo()!=null)
            Caminos(arbol.getHijoIzquierdo(),ref camino,ref caminos,ref copia);
        if(arbol.getHijoDerecho()!=null)
            Caminos(arbol.getHijoDerecho(),ref camino,ref caminos,ref copia);
        camino.RemoveAt(camino.Count - 1);
        return camino;
    }
    return caminos;
}
```

Es una función de tipo Lista de arbol binario de decisión que recibe como parámetro el árbol binario, la lista de camino, caminos y copia las recibe por referencia y su funcionamiento consta de enlistar y almacenar todos los caminos posibles desde la raíz hasta las hojas.

Esto lo logra chequeando que la raíz no sea null y agregando a camino el árbol.

Luego, verifica que el árbol no sea una hoja, guarda camino en copia y copia camino en caminos, una vez copiado, elimina el último elemento almacenado en camino y retorna el mismo.

Si no lo retorna el programa dará error ya que indica un índice negativo.

Cuando termina, procede a consultar si los hijos, tanto el izquierdo como el derecho, no están vacíos, si la respuesta es que si, si no están vacíos, la función se llama de manera recursiva reemplazando el parámetro arbol por el hijo izquierdo y derecho.

De nuevo elimina el último elemento de camino y lo retorna.

Para finalizar la función, retorna la lista de caminos.

Función **Consulta3**

```
public String Consulta3(ArbolBinario<DecisionData> arbol)
{
    int contadorNivel = 0;
    ArbolBinario<DecisionData> arbolaux;
    c.encolar(arbol);
    c.encolar(null);
    string info = "", infoAux= "";
    while(!c.esVacia())
    {
        arbolaux=c.desencolar();
        if(arbolaux == null)
        {
            if(!c.esVacia())
                c.encolar(null);
            info = info + "Nivel "+ contadorNivel++ + ": " + infoAux + "\n";
            infoAux=""; //vuelvo a almacenar vacio
        }
        else
        {
            infoAux = infoAux + arbolaux.getDatoRaiz().ToString() + " | ";
            if(arbolaux.getHijoIzquierdo()!=null)
                c.encolar(arbolaux.getHijoIzquierdo());
            if(arbolaux.getHijoDerecho()!=null)
                c.encolar(arbolaux.getHijoDerecho());
        }
    }
    return  info;
}
```

En esta función se declaró un nuevo objeto de tipo ArbolBinario<DecisionData> arbolaux.

Se inicia encolando a arbol en la cola y luego un null, ya que sirve para separar por niveles.

Se declararon las variables info e infoAux del tipo string en las cuales se almacenarán los datos de la raíz, los hijos del árbol y los niveles del mismo.

Se instancia un bucle while cuya condición de corte es que la cola esté vacía, mientras esto no pase, arbolaux almacenará lo que desencole la cola.

Seguido, pregunta si arbolaux es null y pregunta si la cola no está vacía, si no lo está, encola null y sigue con la línea de ejecución la cual es guardar en info el valor de info concatenado el nivel y el valor de infoAux, luego, vuelve a igualar infoAux a vacío.

Si la cola está vacía almacena en infoAux el valor de infoAux concatenado el valor de la raíz y un separador.

Luego, pregunta si el hijo izquierdo y derecho no están vacíos y encola el contenido de hijo izquierdo y derecho del árbol auxiliar en sus casos respectivamente.

Para finalizar, retorna la variable info que es la que contiene los datos separados por niveles.

Diagrama UML de las clases más importantes involucradas en las especificaciones del sistema.

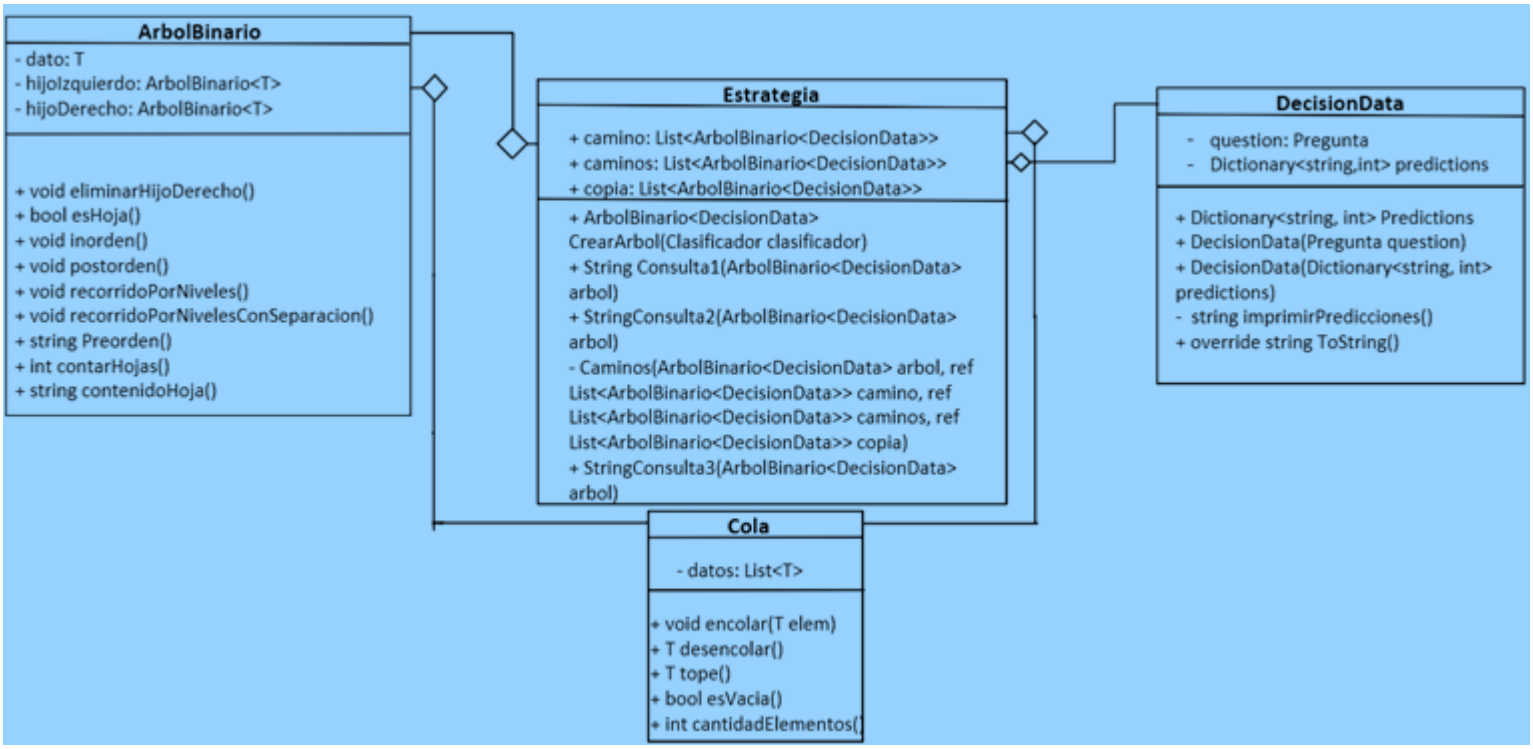


Diagrama UML

Ejecución

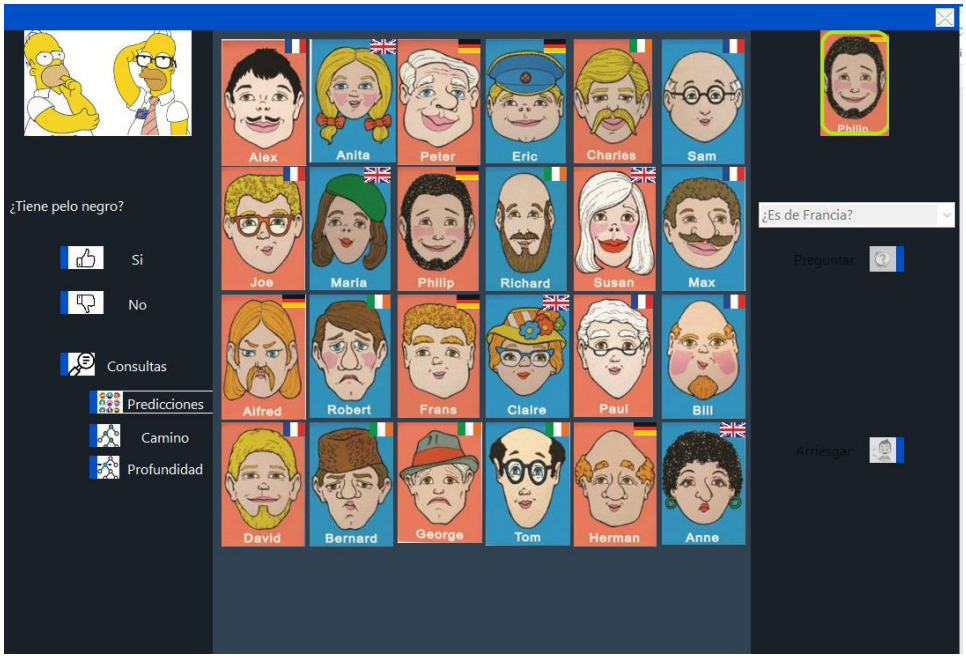
Para comenzar a jugar, lo primero que se hará será cargar el directorio de la carpeta dataset en la pantalla de inicio del juego y click en iniciar.

Pantalla de inicio

Una vez iniciado el juego, se elegirá el personaje con el que se va a jugar

Elección de personaje

A continuación, esta será la pantalla que se mostrará, donde se pueden ver las preguntas que hace la máquina con respecto al personaje que fue seleccionado, las opciones de si o no para que se responda, las consultas que se pueden hacer, la opción de elegir qué preguntas hacerle sobre su personaje a la máquina y la opción de arriesgar.



Pantalla inicial

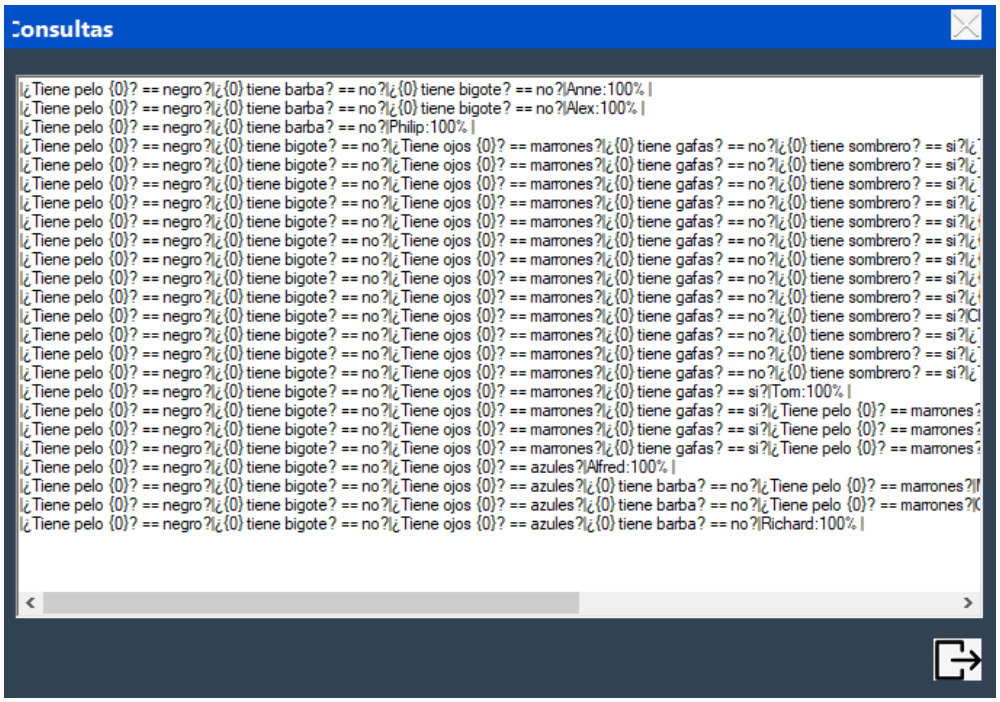
Las consultas son las siguientes:

Consulta Predicciones

Predicciones

Esta consulta lo que muestra es el porcentaje de predicción de los posibles personajes que pueden llegar a ser según las preguntas que la máquina vaya haciendo, esta lista de personajes se va reduciendo pregunta tras pregunta hasta que llega a un nombre del personaje elegido.

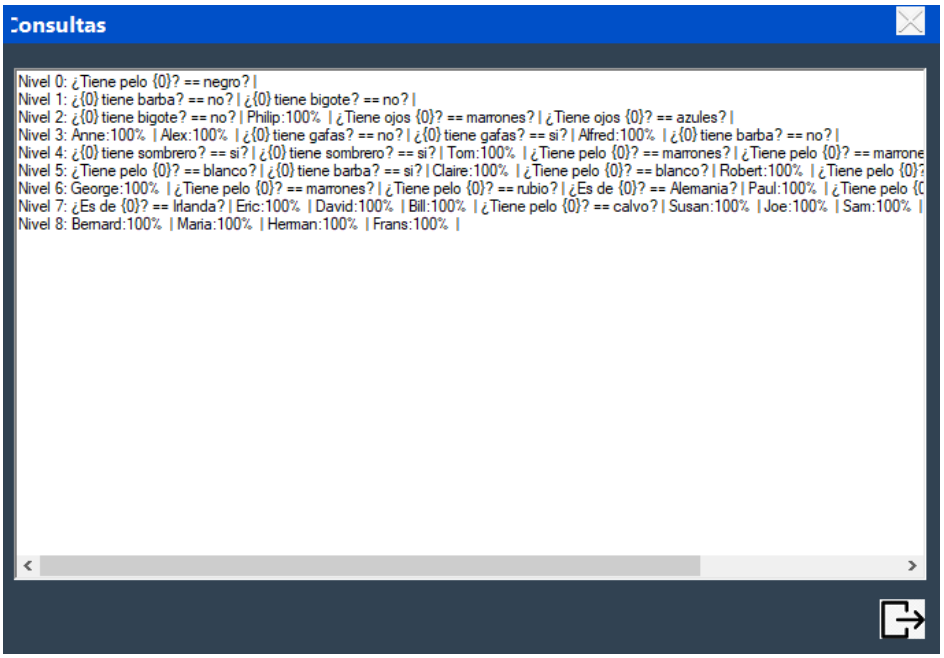
Consulta Camino



Caminos

En esta consulta se pueden observar todos los posibles caminos hacia una predicción, muestra las preguntas que llevan hacia cada uno de los personajes, y, a medida que avanza el juego estos se van reduciendo.

Consulta Profundidad

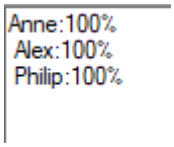


Profundidad

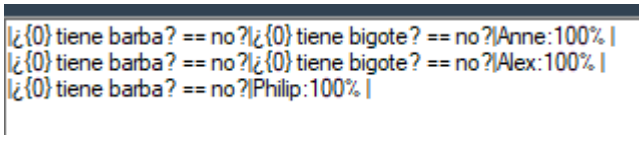
En esta consulta lo que se muestra son los niveles del árbol de decisión desde la raíz hasta las hojas.

A modo de ejemplo se desarrollará la explicación del juego con Phillip, esto valdrá para todos los personajes que sean seleccionados al principio del juego.

Cuando comienza el juego la máquina hace la primera pregunta *"¿Tiene pelo negro?"* el personaje elegido si lo tiene, entonces, se respondió *"Si"* por lo que generó una nueva lista de predicciones en la cual se observa el nombre del personaje seleccionado. Como se dijo anteriormente en la explicación de *Consulta Predicciones*, a medida que avanza con las preguntas esta lista se va reduciendo hasta llegar a adivinar el personaje.



Lista de predicciones Consulta1



Caminos Consulta2

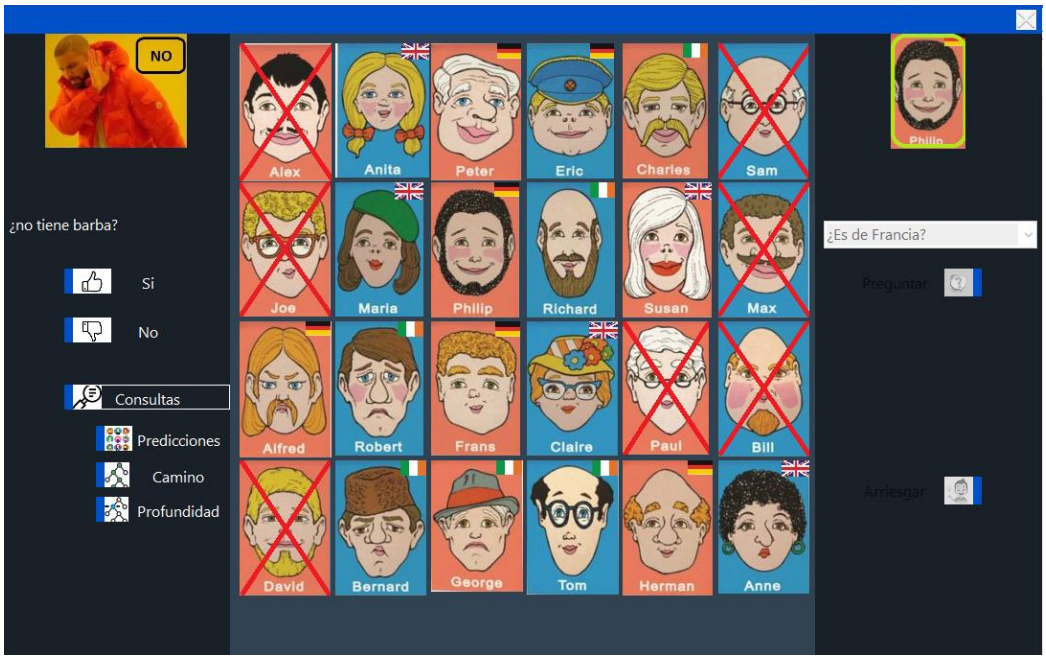
Nivel 0: ¿{0} tiene barba? == no? |
Nivel 1: ¿{0} tiene bigote? == no? | Philip:100% |
Nivel 2: Anne:100% | Alex:100% |

Profundidad Consulta3

Cuando se responde la pregunta que hace la máquina, toca que se pregunte a la misma sobre su personaje con las preguntas que se muestran a continuación

Lista de preguntas

Se preguntó si su personaje era de Francia y respondió que no, entonces, se procede a tachar a los que provienen de ese país.



Respuesta de la máquina

Luego, genera una nueva pregunta "*¿No tiene barba?",* el personaje si tiene, entonces se responde "*No*", como en su lista de predicciones sólo hay un personaje con estas características automáticamente da la predicción y finaliza el juego.

Predicción

Opción de arriesgar

Si a medida que se le va haciendo las preguntas a la máquina se puede deducir el personaje que eligió, está la opción de arriesgar y funciona de la siguiente manera

Se presiona la opción Arriesgar, se selecciona el personaje

Opción arriesgar.

Una vez seleccionado, se presiona Aceptar y aparecerá un mensaje para avisar si la elección fue la correcta o no.

En caso de ser correcta o incorrecta se mostrará lo siguiente

Opción correcta

Opción incorrecta

Se puede volver al inicio o finalizar el juego.

Conclusiones

Si bien se tuvieron adversidades se logró el objetivo final, el cual consiste en aprender más sobre el funcionamiento de los árboles binarios y su implementación en los diferentes campos de la tecnología. También así, el funcionamiento del proceso de hallar caminos y los recorridos de estos para obtener información de diferentes maneras.