Práctica Final: Jugador automático para "¿Quién es quien?"

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. Introducción

El objetivo de esta práctica es utilizar un TDA Árbol Binario (TDA bintree) para implementar un jugador automático para el juego "¿Quién es quién?".

Los requisitos para poder realizarla son:

- Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos.
- Haber estudiado el Tema 2: Abstracción de datos. Plantillas.
- Haber estudiado el Tema 3: T.D.A. Lineales.
- Haber estudiado el Tema 4: STL e iteradores.
- Haber estudiado estructuras de datos jerárquicas: Árboles.

2. El problema: Juego del "¿Quién es quién?"

¿Quién es quién?" (en inglés "Guess who?"), es un juego de adivinanzas para dos jugadores, concebido y fabricado por primera vez por Milton Bradley en los años 80 .

En este juego, cada jugador dispone de un tablero idéntico que contiene 24 dibujos de personajes identificados por su nombre. El juego empieza al seleccionar cada jugador una carta al azar de una pila separada de cartas, que contiene las mismas 24 imágenes. El objetivo del juego consiste en ser el primero en determinar qué carta seleccionó el oponente. Esto se consigue haciendo preguntas, una por turno, cuya respuesta puede ser sí o no, para eliminar candidatos.

Un ejemplo de pregunta es «*¿Tu personaje tiene bigote?*». La respuesta de nuestro oponente nos permitirá eliminar los personajes que no cumplan el criterio, tumbando estas tarjetas del tablero.

El estudiante debe implementar un jugador automático para jugar al "¿Quién es quién?". En particular, el jugador automático debe hacer las preguntas, recoger la respuesta "Si"/"No" del jugador humano y continuar haciendo preguntas hasta deducir el personaje con el que el jugador humano está jugando.

3. Definición del tablero de personajes

Para poder jugar al juego "¿Quién es quién?" primero definiremos un tablero de personajes utilizando un formato de texto plano que se indica a continuación:

Atributo1 \t **Atributo2** \t … \t **AtributoK** \t **Nombrepersonaje** \n
v_{11} \t v_{12} \t … \t v_{1k} \t p_1 \n
v_{21} \t v_{22} \t … \t v_{2k} \t p_2 \n
……
……
v_{N1} \t v_{N2} \t … \t v_{Nk} \t p_N \n

Este fichero define un tablero de N personajes y k atributos diferentes, donde v_{ij} toma un valor en {0, 1} (0="No", 1= "Si"), indicando si el personaje i, de nombre p, presenta o no el atributo j. El número máximo de personajes que pueden definirse con k atributos binarios es 2^k. Para que un tablero sea válido, es obligatorio que dos personajes distintos tengan, al menos, un atributo distinto.

Por ejemplo, el siguiente tablero (Tabla 1) define 5 personajes diferentes utilizando 4 atributos (N=5, k=4):

Mujer	Ojos marrones	Pelo castaño	Tiene gafas	Nombre del personaje
0	1	1	1	Ernesto
1	0	0	0	Ana
0	0	1	1	Juan
0	1	0	0	Antonio
1	1	1	0	Pilar

Tabla 1. Ejemplo de tablero.

Este fichero con la definición del tablero de personajes debe ser conocido por parte del jugador humano y del jugador automático, ya que el juego se desarrollará partiendo de esta descripción de personajes.

4. Jugador automático para "¿Quién es quién?"

Para implementar un jugador automático que juegue a "¿Quién es quién?", el estudiante completará la clase QuienEsQuien (que se proporciona en el material adjunto a la práctica) para la que se propone la siguiente representación:

```
class QuienEsQuien{
private:
    vector<string> personajes;     // nombres de los personajes
    vector<string> atributos;       // nombres de los atributos
    vector<vector<bool>> tablero; // tablero de características para cada personaje
                                  // tamaño: size(personajes) x size(atributos)
    bintree<Pregunta> arbol;      // árbol de preguntas para adivinar personajes
     bintree<Pregunta>::node jugada_actual;      // jugada actual
    vector<string> personajes_images;   // nombres de las imágenes de  personajes en disco
                                        //(para jugar en modo grafico)
    TableroGrafico *tg;          //puntero a un tablero grafico(para jugar en modo gráfico)
   consola *con;                //puntero a un consola grafica (para jugar en modo gráfico)
   string imagen_ocultar;       //nombre de la imagen en disco para ocultar un personaje en
                                //  el tablero grafico (para jugar en modo gráfico)
   bool modo_graph;             // flag que indica si se juega en modo gráfico o
                                // texto
   ...

};
```

En esta sección se describen cada uno de los elementos considerados en esta representación, así como algunos de los métodos de la clase. Toda la información está en la especificación de la clase QuienEsQuien (carpeta include/quienesquien.h del material proporcionado).

4.1. Lectura del tablero

La lectura de los ficheros de tablero (como el representado en la Tabla 1) está ya implementada en el método:

```
friend istream& operator >> (istream& is,QuienEsQuien &quienEsQuien);
```

de la clase QuienEsQuien. Este método carga los datos del tablero en las variables tablero, personajes y atributos del objeto de clase QuienEsQuien. Puedes consultar su especificación e implementación en los ficheros quienesquien.h y quienesquien.cpp.

4.2. Representación de las preguntas para adivinar todos los personajes del tablero

Para implementar un jugador automático que juegue a "¿Quién es quién?", el estudiante construirá un *árbol binario* para representar la sucesión de preguntas necesaria para adivinar cada personaje del tablero. Este árbol binario se almacena en el atributo arbol de la clase QuienEsQuien. Para ello, el campo etiqueta de cada nodo n de este árbol almacenará elementos de tipo Pregunta, definido como:

```
class Pregunta{
       string atributo;      // atributo sobre el que se pregunta en este nodo
       int num_personajes;  // número de personajes que quedan en el tablero
};
```

El nodo raíz del árbol codifica la primera pregunta que formula el jugador automático. Los nodos hijos del nodo raíz (nodos de nivel 2) representan la segunda pregunta que formula el jugador automático, y así sucesivamente. La Figura 1 muestra un esquema de esta representación aplicada a la Tabla 1.

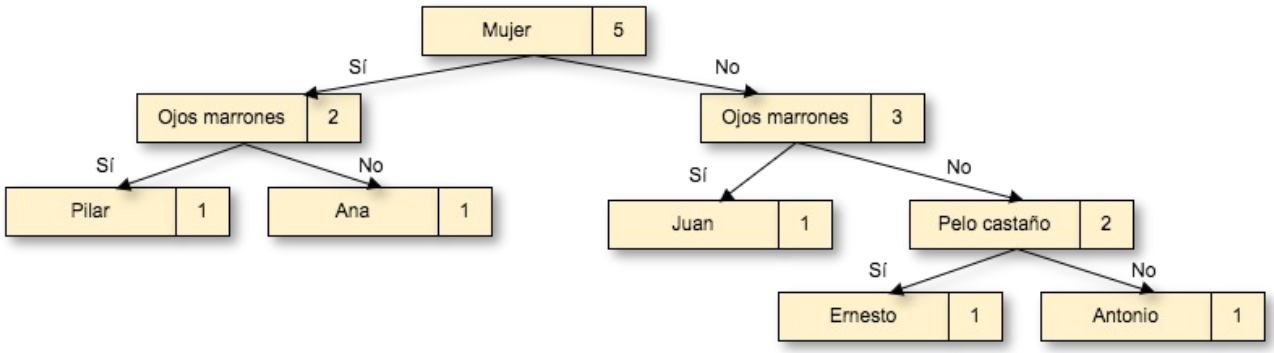


Figura 1. Árbol de preguntas asociadas a Tablero 1. El árbol se ha construido formulando preguntas sobre los atributos de Tablero 1 según el orden en el que vienen dados en el fichero

Cada nodo n del árbol codifica una pregunta asociada al juego y el número de personajes que aún no han sido eliminados. El hijo izquierda de n representa el tablero de personajes que tienen respuesta "Si" a la pregunta formulada en n, y el hijo derecha el tablero de personajes que se obtiene con la respuesta "No" a la pregunta. Los nodos hoja del árbol codifican un tablero en el que sólo queda un personaje que no ha sido descartado. En este caso, el campo atributo debe contener el nombre del personaje en cuestión y el campo num_personajes es igual a 1.

Por tanto, el camino desde la raíz del árbol hasta n determina la sucesión de preguntas y respuestas obtenidas hasta llegar a ese tablero, y los nodos hoja descendientes de n determinan los personajes aún no tumbados en el tablero asociado a n.

4.3. Proceso de juego de una partida de "¿Quién es Quién?"

Una vez construido este árbol con sucesiones de preguntas para identificar cada personaje, una partida del juego se implementa como un recorrido por este árbol, comenzando desde la raíz y hasta alcanzar un nodo hoja. De este modo, durante el transcurso de la partida, será necesario registrar el nodo del árbol por el que va la partida. Este es el uso que daremos al atributo jugada_actual de la clase QuienEsQuien. Al inicio de la partida, jugada_actual se inicializa con el nodo raíz del árbol. El jugador automático formula la pregunta asociada a este nodo y la respuesta del usuario determina el camino a seguir (Sí -> hijo izquierda, No -> hijo derecha). Este proceso continúa hasta que jugada_actual alcanza un nodo hoja. En ese momento, el jugador automático ya conoce la respuesta, y la imprime en pantalla: "*¡Ya lo sé! tu personaje es X*" donde X se reemplaza por el nombre del personaje almacenado en la etiqueta del nodo hoja. Con esto acaba la partida, jugada_actual se reinicia al nodo raíz del árbol y se da al usuario la opción de jugar otra vez.

4.4. Construcción del árbol de preguntas

Construcción básica.

Para implementar un jugador automático básico para el juego del "Quién es quién", construiremos el árbol binario descrito en la sección 4.2 a partir de la información almacenada en los atributos tablero, personajes y atributos. La forma más sencilla de construir el árbol es formular la pregunta asociada al atributo i (1<= i <=size(atributos)) en todas las ramas de nivel i del árbol binario (excepto las hojas), tal y como se muestra en la Figura 1. Es decir, en el primer nivel (nodo raíz) formulamos la pregunta asociada al atributo que ocupa la primera columna del fichero; en el segundo nivel, la pregunta asociada al segundo atributo, y así sucesivamente hasta llegar a los nodos hoja. Nótese que con esta propuesta se hacen siempre las mismas preguntas y en el mismo orden, sean cuales sean las respuestas recibidas (es decir, para todas las ramas del árbol hasta llegar a cada hoja).

Omitir preguntas inútiles.

La construcción básica puede llevar a situaciones donde se haga una pregunta cuya respuesta sea la misma para todos los personajes que aún no han sido tumbados, por lo que esta pregunta podría ser omitida. Por ejemplo, si en el nivel 1 se pregunta "*¿es mujer?*", no tendría sentido preguntar a continuación: "*¿es hombre?*". Esta situación se muestra en la Tabla 2 y gráficamente en la Figura 2.

Mujer	Hombre	Ojos marrones	Ojos negros	Pelo castaño	Nombre del personaje
1	0	1	0	1	Pilar
1	0	0	1	1	Ana
0	1	1	0	1	Juan
0	1	0	1	1	Ernesto
0	1	0	1	0	Antonio

Tabla 2. Ejemplo de tablero ordenado para identificar preguntas inútiles

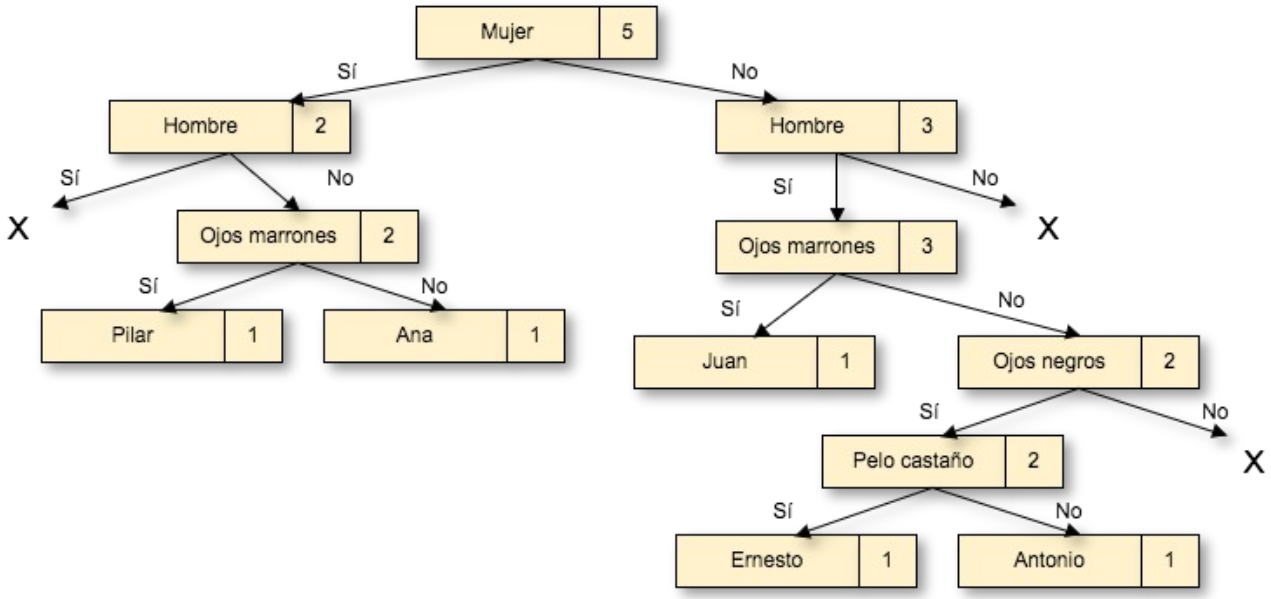


Figura 2. Árbol de partidas asociadas a Tablero 2. El árbol se ha construido formulando preguntas sobre los atributos de Tablero 2 según el orden en el que vienen dados en el fichero.

Se propone una mejora sencilla para omitir preguntas inútiles. Una vez construido el árbol siguiendo las indicaciones de la sección "Construcción Básica", el alumno puede efectuar un recorrido por todos los nodos del árbol, detectando si existe algún nodo no hoja que sólo tenga un hijo. Si se detecta este caso, ese nodo puede ser reemplazado por su hijo (Figura 3). En la práctica, esto supone que se omiten aquellas preguntas cuyas respuestas no discriminan a, al menos, un personaje no tumbado del resto de personajes no tumbados. El resultado se muestra en la Figura 4

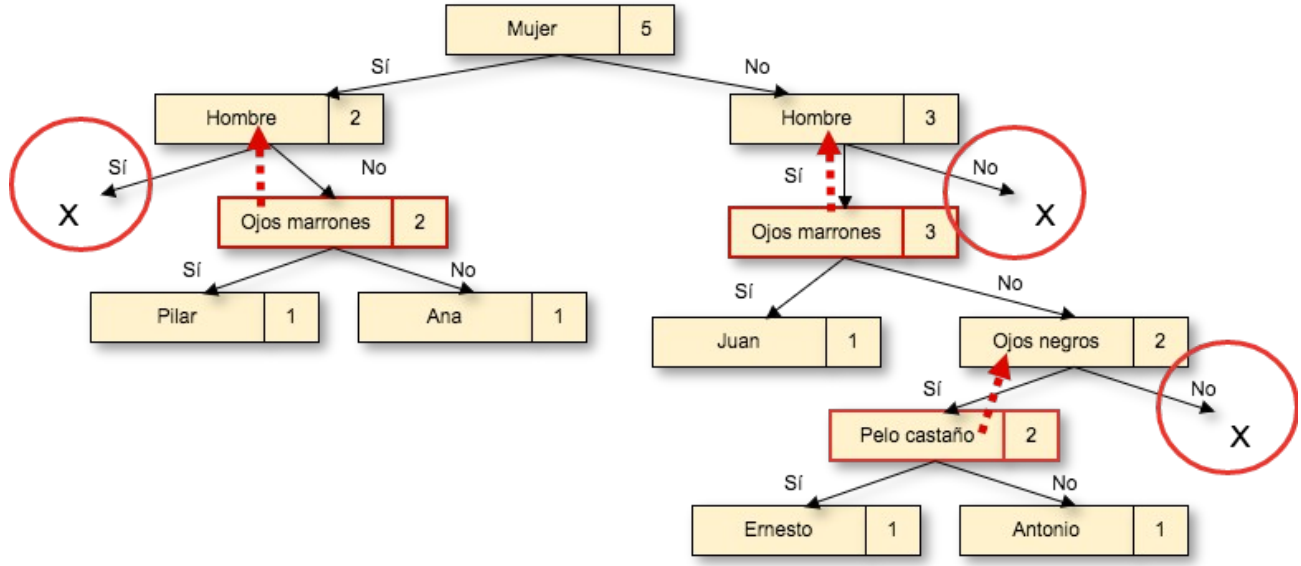


Figura 3. Un recorrido por el árbol binario permite detectar nodos que sólo tienen un hijo. Estos nodos codifican preguntas que podrían omitirse. Para ello, se reemplaza dicho nodo por su hijo.

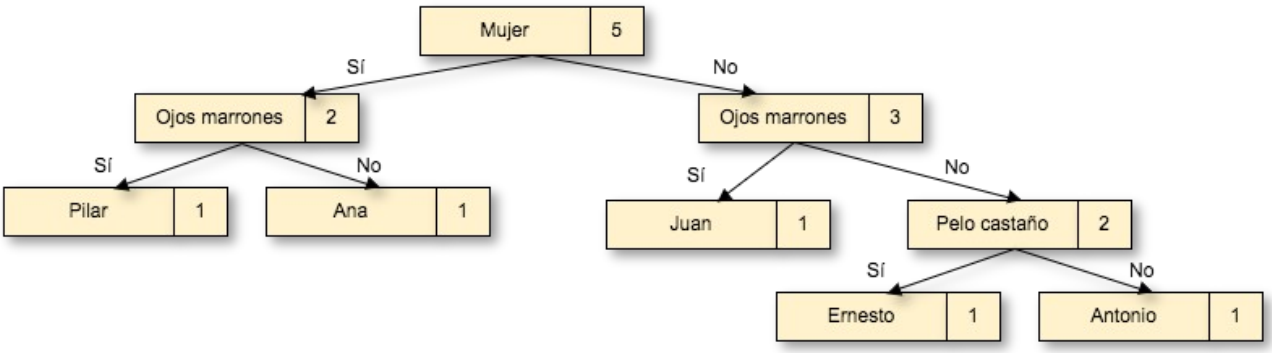


Figura 4. Resultado de la eliminación de nodos no hoja que sólo tienen un hijo. El árbol resultante es más compacto que el de las figuras 2-3.

Mejora: Elegir la pregunta más adecuada.

Una mejora sustancial en la construcción del árbol pasaría por elegir, en cada momento, la pregunta que divide al conjunto de personajes no tumbados en dos subconjuntos de tamaño similar. De este modo, el árbol obtenido está mejor balanceado y tendríamos que formular menos preguntas, en promedio, para resolver partidas sobre cualquier personaje. Esta idea intuitiva es la que subyace bajo el concepto de *Entropía* en Teoría de la Información y la construcción de *Árboles de Decisión* en Inteligencia Artificial. Os invitamos a explorar estos conceptos para implementar una métrica para elegir la mejor pregunta en cada momento.

Para ello, pudiera ser necesario modificar la representación de la clase QuienEsQuien o Pregunta para representar las características de los personajes que quedan sin tumbar en cada momento. La representación y la implementación de estas mejoras queda a vuestra elección. La siguiente sección explica la línea de las mejoras.

Módulo: la inteligencia de la máquina

En este apartado queremos hacer mejor jugador a la máquina. El jugador máquina juega mejor si es más rápido en adivinar el personaje oculto de su contrario. Esta rapidez dependerá de la táctica (heurística) escogida para realizar la pregunta al contrario. Por ejemplo, la táctica más fácil de programar es escoger una de las preguntas, que aún no se ha realizado, de forma aleatoria. Pero existen mejores heurísticas que se proponen al alumno/a que programe de forma voluntaria:

- Escoger la pregunta de mayor entropía. La entropía se maximiza cuando el conjunto original se divide en dos subconjuntos con igual número de personajes. Uno de los conjuntos contestando Sí a la pregunta y otro conjunto contestando No. De forma matemática, la entropía se formula como:

$$H(x) = -\sum p(x_i) \log_2(p(x_i))$$

  donde

  - x es la pregunta,
  - x_i son las posibles respuestas (en nuestro caso son dos: "Sí"o "No") y
  - $p(x_i)$ es la probabilidad de que se dé la respuesta x_i a la pregunta x con un subconjunto de personajes (los no descartados del conjunto de partida).

Para aclarar mejor estos conceptos veamos un ejemplo con mayor detalle. Suponiendo el conjunto

de personajes de la figura 5 entre las siguiente dos preguntas:

1. ¿Es Mujer su Sexo?
2. ¿Es Claro su Color de Ojos?

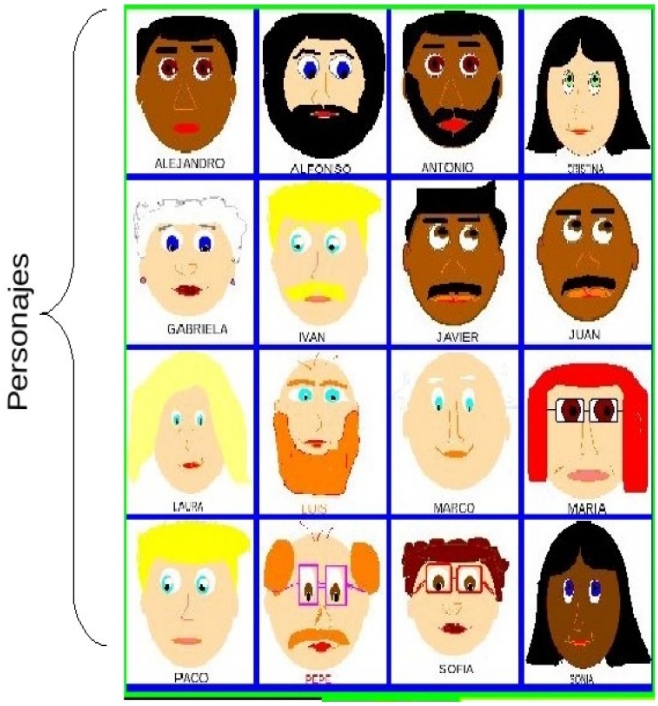


Figura 5. Ejemplo de personajes

Pasamos a calcular la entropía de cada una de las preguntas:

$$H(\text{¿ Es Mujer Su Sexo ?}) = -(p(No) * \log_2(p(No)) + p(Si) * \log_2(p(Si)))$$

donde las probabilidades son:

$$p(No) = \frac{10}{16} \quad \text{y} \quad p(Si) = \frac{6}{16}$$

Si hacemos el cálculo obtenemos que

$$H(\text{¿ Es Mujer Su Sexo ?}) = 0.954434$$

y

$$H(\text{¿ Es Claro su Color de Ojos ?}) = -(p(No) * \log_2(p(No)) + p(Si) * \log_2(p(Si)))$$

donde las probabilidades son:

$$p(Si) = \frac{9}{16} \quad \text{y} \quad p(No) = \frac{7}{16}$$

Haciendo el cálculo de la entropía obtenemos que

$$H(\text{¿ Es Claro su Color de Ojos ?}) = 0.9886$$

Por lo tanto la máxima entropía se obtiene con la pregunta *¿Es Claro su Color de Ojos?*. Intuitivamente, alta entropía quiere decir que nos acercamos más a la equiprobabilidad de

respuestas, o dicho de otra forma: más o menos tenemos el mismo número de respuestas en ambos sentidos. Esto es bueno porque garantiza que, en el peor de los casos, descartamos el mayor número posible de personajes.

Si elegimos preguntas con baja entropía, puede ser que tengamos suerte y descartemos muchos personajes con una pregunta pero si somos poco afortunados descartaremos muy pocos. Si encadenamos varias preguntas "con mala suerte" el resultado será muy malo. Esta es una mala estrategia, ya que lo habitual es pensar siempre en la situación más desfavorable para decidir qué hay que hacer.

Hay que tener en cuenta que no deberíamos volver a realizar preguntas ya hechas. Con esto último se quiere hacer notar que, por ejemplo, si el jugador preguntó *¿Es Claro su Color de Ojos?* y la contestación fue afirmativa, en el futuro no se volverá a hacer la misma pregunta.

4.5. Métodos sobre el árbol de preguntas

Se propone al alumno que implemente los siguientes métodos sobre el árbol de preguntas para la clase QuienEsQuien:

- `QuienEsQuien::crear_arbol ( )`

  Este método construye el árbol de preguntas para todos los personajes del tablero, según lo descrito en las secciones 4.2 y 4.4.

- `QuienEsQuien::iniciar_juego ( )`

  Este método implementa el desarrollo de una partida contra un usuario humano, tal y como se describe en la sección 4.3.

- `QuienEsQuien::informacion_jugada(bintree<Pregunta>::node jugada)`

  Este método se aplica sobre un nodo del árbol de preguntas (jugada) para obtener un set<string> con los nombres de los personajes que aún no han sido tumbados en el tablero.

- `QuienEsQuien::profundidad_promedio_hojas()`

  Este método permite calcular la media de la profundidad de las hojas del árbol. Este valor representa el número (promedio) de preguntas necesarias para adivinar cada personaje. A menor profundidad promedio, mejor solución. Esta métrica es un indicador para evaluar la calidad de vuestra solución.

Métodos adicionales

- `QuienEsQuien::preguntas_formuladas (bintree<pregunta>::node jugada)`

  Este método se aplica sobre un nodo del árbol de preguntas (jugada) para obtener una descripción de las preguntas formuladas anteriormente y las respuestas dadas por el usuario hasta ahora. Por ejemplo:

  "El personaje oculto tiene las siguientes características:
  Mujer – no
  Ojos Marrones – si
  pero aún no sé cuál es".

- `QuienEsQuien::aniade_personaje (string nombre, vector <bool> caracteristicas,string nombre_imagen_personaje="")`

  Dado un árbol ya construido para un tablero, imaginemos que tenemos que añadir un nuevo personaje a partir de su nombre y descripción asociada (vector<bool>), pero sin rehacer el árbol completo. Este método inserta el personaje nuevo en el árbol ya construido. Para ello, es necesario utilizar las características del nuevo personaje para recorrer el árbol desde la raíz y encontrar dónde insertar el nuevo personaje. Típicamente, este recorrido por el árbol nos llevará a un nodo hoja (otro personaje del tablero) y tendremos que añadir un nivel más (una pregunta más) para distinguir al nuevo personaje del nodo hoja que ya estaba en el árbol. Se asume que el vector<bool> de características tiene el mismo tamaño y los atributos están en el mismo orden que el especificado en el fichero de entrada.

- `QuienEsQuien::elimina_personaje (string nombre)`
  Este método elimina un personaje del árbol ya construido, siguiendo un proceso similar al descrito en el método anterior: utilizaremos las características del personaje para recorrer el árbol desde la raíz y encontrar su nodo hoja asociado. Al eliminarlo, típicamente tendremos que eliminar también a su padre (porque ahora será una pregunta inútil), o mejor dicho, reemplazar a su padre por su otro nodo hijo.

- **Jugar en modo gráfico**
  El estudiante puede replicar el juego en modo gráfico. Para ello dispone de las clases tablerografico, ventana y consola. Tablerografico permitirá visualizar los personajes aún activos y con la consola podrá dar la respuesta a las preguntas formuladas por la máquina. Tanto tablerografico como consola se integran en una ventana.

5. El programa QuienEsQuien

El programa QuienEsQuien tendrá diferentes formas de ejecutarse desde la línea de órdenes:

- Jugar en modo texto. Un ejemplo para ejecutar desde la línea de órdenes será:

```
bin/quienesquien datos/personajes-1.csv
```

- Jugar en modo gráfico (opcional). Un ejemplo para ejecutar desde la línea de órdenes será:

```
bin/quienesquien datos/personajes_graficos.csv datos/Tapada.png
```

- Generar tableros aleatorios. Un ejemplo para ejecutar desde la línea de órdenes será:

```
bin/quienesquien aleatorio
```

- Limpiar el árbol de preguntas que no divide al conjunto de personajes. Un ejemplo para ejecutar desde la línea de órdenes será:

```
bin/quienesquien datos/personajes-1.csv limpiar
```

6. Instrucciones de entrega

Para la implementación de esta práctica, **se proporciona** el siguiente material:

- Clases bintree y node: implementación del TDA bintree.
- Clase QuienEsQuien: almacena una representación del tablero, los personajes y sus

atributos, además de un bintree con las preguntas necesarias para adivinar todos los personajes de un tablero dado. Esta clase está incompleta.

- Clase Pregunta: representa un nodo del árbol de preguntas de QuienEsQuien.
- Programa principal main.cpp: implementa un programa que lee un fichero con una configuración de tablero y crea un objeto QuienEsQuien para que el usuario juegue contra un jugador automático.
- Ficheros con tableros de ejemplo (carpeta /datos)
- Método tablero_aleatorio en la clase QuienEsQuien para generar tableros aleatorios de tamaño dado.
- CmakeLists.txt: para llevar a cabo la compilación.

La práctica tiene una puntuación total de 1 punto. Para obtener esta puntuación, **se pide**:

- Completar la implementación de la clase QuienEsQuien, en particular, de los métodos descritos en la sección 4.5 (sin métodos adicionales). La correcta implementación de estos métodos permitirá obtener una puntuación de 0.5 puntos en la nota de la práctica.
- Completar la implementación de las ″Mejoras" y "Métodos Adicionales", propuestos en las secciones 4.4 y 4.5. La correcta implementación de estos métodos permitirá obtener una puntuación de 0,5 puntos en la nota de la práctica.
- Entregar una memoria donde se describa la solución propuesta y cualquier mejora, en su caso, que haya sido implementada para construir el árbol de preguntas.

Normas de la entrega

El alumno/a deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "submission.zip" y entregarlo antes de la fecha indicada en la plataforma de docencia virtual. No se incluirán ficheros objeto, ni ejecutables. Es recomendable hacer una limpieza para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

Se dispone de la siguiente estructura de ficheros:

```
./
  estudiante/     (Aquí irá todo lo que desarrolle el estudiante)
    datos/ (ejemplos de ficheros de personajes, tableros, imágenes de personajes)
    doc/          (Imágenes y documentos extra para Doxygen)
    include/      (Archivos cabecera)
    src/          (Archivos fuente)
    CMakeList.txt (Instrucciones Cmake si se quiere usar)
    Doxyfile.in   (Archivo de configuración de Doxygen)
```

- Es importantísimo (si se va a usar) leer detenidamente y entender qué hace CmakeList.txt.
- El archivo Doxyfile.in no tendríais porqué tocarlo para un uso básico, pero está a vuestra disposición por si queréis añadir alguna variable (no cambiéis las existentes).
- Crear un archivo submission.zip que incluya TODO lo necesario cara a subirlo a Prado para la entrega.

7. Referencias

- Rosa Rodriguez-Sánchez, J. Fdez-Valdivia, J.A. García, Estructuras de Datos en C++. Un enfoque práctico, 2020.
- Rosa M. Rodríguez Sánchez. Quién es Quién. Documento práctico, 2015.
- Garrido, A. Fdez-Valdivia, J. "Abstracción y estructuras de datos en C++". Delta publicaciones, 2006.

Anexo

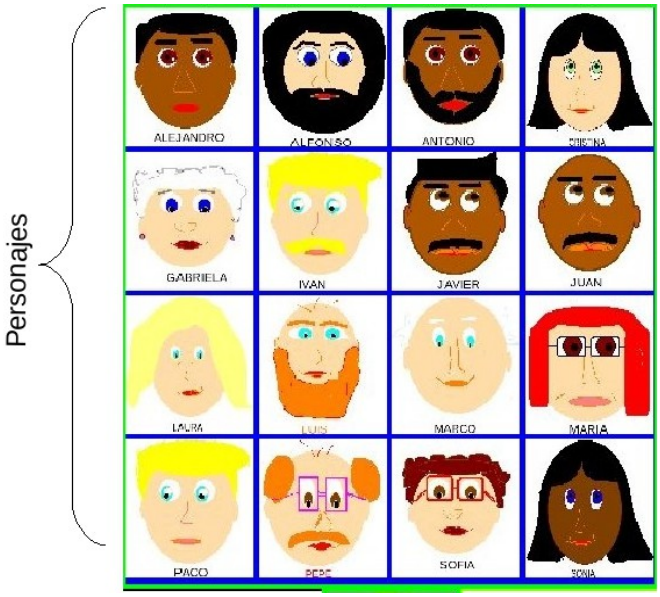
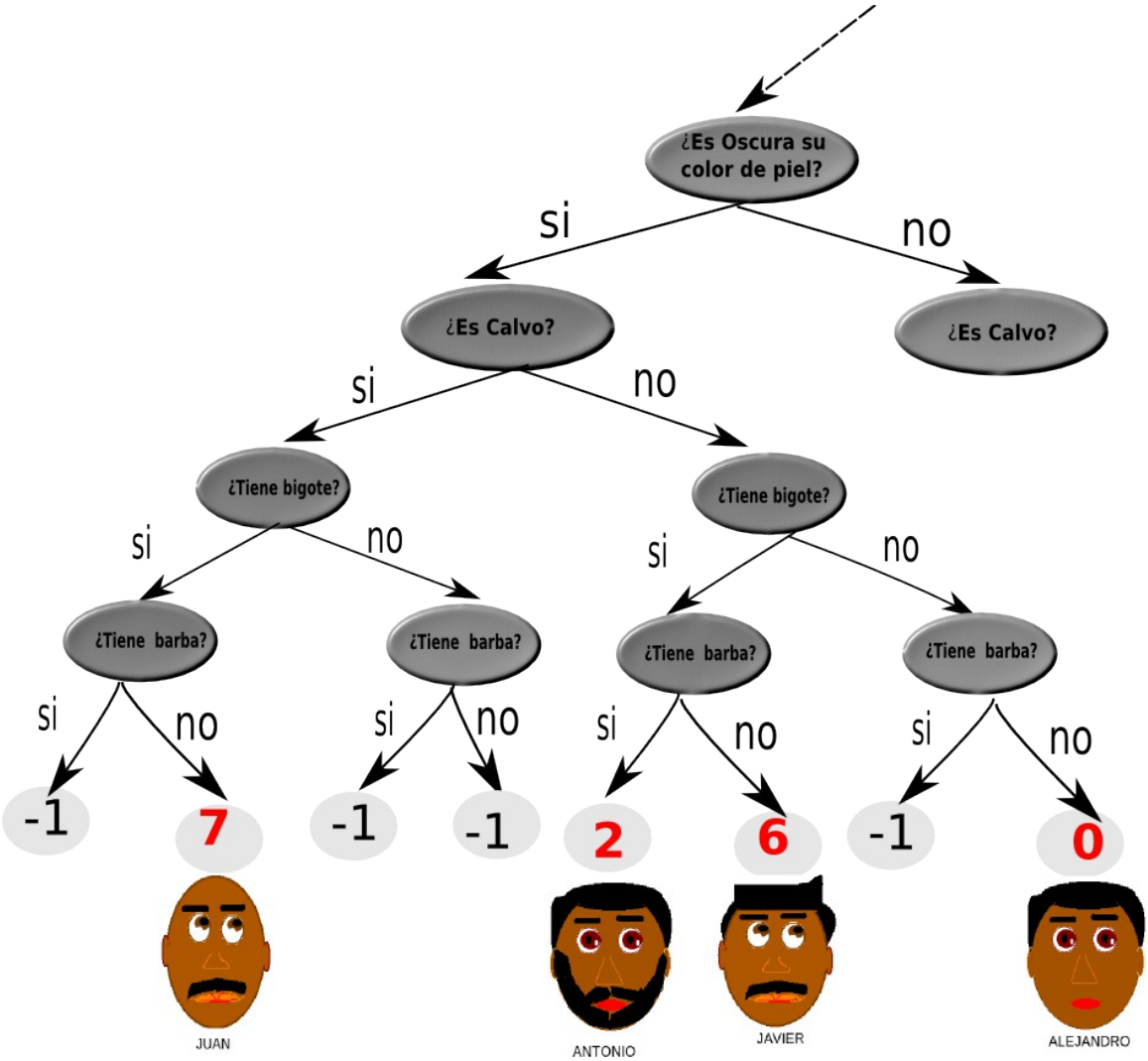


Imagen de posibles personajes del juego



Parte del árbol que codifica los rasgos de los personajes en la Figura anterior

```
Personaje: Alfonso
Preguntas y contestaciones son:
¿Es Corto su pelo?Si
¿Es Largo su pelo?No
¿Es Hombre?Si
¿Es Mujer?No
¿Tiene gafas?No
¿Tiene la piel Oscura?No
¿Tiene la piel Clara?Si
¿Tiene el pelo Marron?No
¿Tiene el pelo Pelirrojo?No
¿Tiene el pelo Blanco?No
¿Tiene el pelo Rubio?No
¿Tiene el pelo Negro?Si
¿Es Oscuro su color de ojos?No
¿Es Claro su color de ojos?Si
¿Es calvo?No
¿Tiene bigote?Si
¿Tiene barba?Si
Codigo asociado 11010100001000101
…
```

Código binario asociado a un personaje concreto