Como se Hizo...

Quien es Quien

Carlos de la Torre

Miguel Martínez Castellanos

Introducción.

En esta práctica se nos plantea la realización de una práctica en la que simularemos un juego entre dos adversarios, que a través de turnos se efectuarán preguntas de respuestas si o no(0 ó 1)para ir descartando personajes y así obtener el personaje que le fue asignado el comienzo del juego.

Para ellos nos han facilitado varios ficheros de configuración en los que se encuentras los datos organizados de manera ordenada separados con el carácter “#”.Además tenemos unos ficheros para los que se podrá comprobar la efectividad del programa con un caso de banderas y otros dos casos de caras de personajes en los que mediante las preguntas que hemos mencionado anteriormente se irán descartando posibilidades hasta encontrar a nuestro “quien es quien”.

Objetivos de la Práctica.

Para la realización hemos llevado a cabo un análisis de las necesidades de la implementación y optamos por crear las siguientes estructuras de datos que se citan a continuación.

En primer lugar definiremos las clases creadas y seguidamente el TDA utilizado en cada una de ellas.

- $Class\ Persona$: Esta clase se creó para definir un nuevo tipo de dato abstracto y así tratar a cada uno de los participantes con los siguientes atributos.
  - Un dato de valor entero “num”para poder diferenciarlo de otro jugador.
  - Un dato Nombre que será el nombre del personaje al que representa con un string.
  - Un dato Path_photo que será una cadena de string que contenga la ruta donde se encuentra la imagen del personaje.
  - Un dato Código que es una cadena de string de 0s y 1s que indican las respuestas que se dan a las diferentes preguntas, 0 = NO, 1 = SI.

- ***Función de abstracción:***

  La función de abstracción de esta clase es sencilla puesto que suponiendo que p es un objeto de tipo Persona podemos mostrar el contenido de sus campo sin ningún problema.

  fA : rep −−→ T.D.A. especificación.

  Donde p es una instancia de objeto abstracto de tipo “Persona” que sirve para crear T.D.A.

fA : rep −−> p.num, p.Nombre, p.Path_photo, p.Codigo

- ***Invariante de la representación:***

  El campo num solo podrá recibir valores positivos hasta n personajes.
  El dato “Nombre” será una cadena de caracteres para la identificación del jugador.
  El dato “Path_photo” será una cadena de caracteres donde son indicara la ruta de la ubicación de su imagen.
  El dato código será un string compuesto de 0 y 1 para la identificación de camino realizado a través de las preguntas.

- *Class Personas*: Esta clase se creó para definir un nuevo tipo de dato abstracto con el que tendremos una base de datos de personas por el que decidimos usa un map.
  - Un map de primer elemento un entero y de segundo elemento un TDA de Persona.
  - Esta clase tiene un iterador implementado para poder manejar mejor los elementos del map.

  - ***Función de abstracción:***

    La función de abstracción para un map es tan sencillo como acceder a los datos de cada uno de los elementos de acaba objeto del map.

    fA : rep −−> T.D.A. especificación.

    Donde pe es una instancia de objeto abstracto de tipo “Personas” que sirve para crear T.D.A.

    fA : rep −−> pe.first (siendo >=0…n), pe.second(Persona != vacio)

  - ***Invariante de la representación:***

    El primer elemento del map no podrá ser negativo y el segundo elemento será objeto del TDA Persona. (En un map no pueden existir elementos repetidos.)

- $Class\ Jugador$: Esta clase se creó para definir un nuevo tipo de dato abstracto y así tratar a cada uno de los participantes con los siguientes atributos.
  - Un dato tipo string para guardar el nombre.
  - Un dato de tipo Persona para tener el personaje.
  - Un dato de tipo Personas en el que tendrá los posibles personajes.
  - Un dato de tipo Preguntas en la que estarán almacenadas las posibles preguntas que se puedan realizar durante el juego.

- ***Función de abstracción:***

  La función de abstracción de la clase Jugador tendrá que cumplir que para cada jugador este tendrá un nombre asignado , un personaje que el contrincante deberá adivinar y una serie de preguntas disponibles para efectuar durante el juego.

  fA : rep −−→ T.D.A. especificación.

  Donde j es una instancia de objeto abstracto de tipo “Jugador” que sirve para crear T.D.A.

  fA : rep −−→ j.Nombre(siendo != vacio) ,j.personaje(siendo != vacio && ϵ en posible_personajes) , j.posibles_personajes(siendo != vacio) , j.posibles_preguntas(siendo != vacio).

- ***Invariante de la representación:***

  El campo de nombre no podrá estar vacío y será un string.
  El campo que pertenece a Persona guardara todo lo necesario para el personaje que tenga que encontrar el contrincante y este tiene que existir como personaje y reunirá características sobre las preguntas que le puedan hacer.
  El valor de las posibles_personajes será la lista de posibles personajes que haya en el juego y este no estar vacío y enumero máximo de personaje será el indicado por el fichero de los personajes que nos han pasado.
  Posibles_preguntas serán las mismas preguntas para todos los personajes y serán n preguntas indicadas en el fichero inicial.

- $Class\ Preguntas$: Esta clase es una TDA para almacenar las preguntas leídas de fichero.

- Para almacenar las preguntas utilizaremos un map donde el primer campo será el id de la pregunta y el segundo campo un string para la pregunta.
- Esta clase tiene un iterador implementado para poder manejar mejor los elementos del map.

- ***Función de abstracción:***

  La función de abstracción para esta clase solo es necesario tener en cuenta los acceso a los datos ya que en ningún momento de modificarían las preguntas.

  fA : rep −−→ T.D.A. especificación.

  Donde pre es una instancia de objeto abstracto de tipo “Preguntas” que sirve para crear T.D.A.

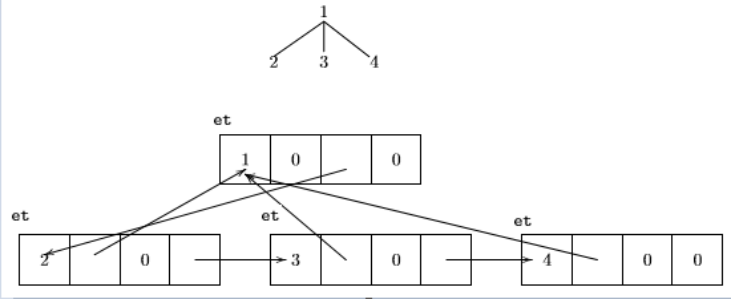
  fA : rep −−→ pre.first(siendo >=0…n), p.second(Pregunta != vacío)

- ***Invariante de la representación:***

  Al ser un map de entero el primer campo no puede tener ningún otro valor por debajo de 0 o superior al número de preguntas.(En un map no puede existir elementos repetidos.)y como segundo campo será la pregunta correspondiente a dicho id.

- $Class\ ArbolGeneral$: Esta clase es una TDA para las relaciones entre un nodo raíz, padre, hijomasalaizquierda y hermanoaladerecha.Para establecer estas relaciones necesitamos de los siguientes tipos de datos.
  - Creamos un struct llamado nodo para recoger todo elemento de cada relación y así poder organizar estos nodos en el árbolgeneral.
  - Esta clase tiene el iterador implementado para poder manejar mejor los elementos del árbol y además la clase del iterador constante.
  - Hemos separado la clase de ArbolGeneral en varios ficheros para trabajar la simplicidad y la limpieza de dicha práctica.
  - Representación:

```
template <class T>
struct info_nodo {
      T et;
      info_nodo<T> * padre, * hijoizq, * hermanodcha;
      info_nodo() {
            padre = hijoizq = hermanodcha = 0;
      }
      infonodo(const T & e) {
            et = e;
            padre = hijoizq = hermanodcha = 0;
      }
}
```



- ***Función de abstracción:***

  La función de abstracción de un árbol general A contiene un nodo distinguido V0 llamado raíz de A y los nodos restantes de A forman un conjunto de árboles A1,A2,…,An

  fA : rep −−→ T.D.A. árbol general .

  Sea V → conjunto de vértices o nodos y A → relación de aristas entre los elementos de V.

  (A,V0) es un árbol si y solo si:

  1. Existe un único vértice V0 ϵ V tal que V0 no tiene ninguna entrada v0 el llamado raíz del árbol A.
  2. Para todo v E V tal que v=/ v0,v tiene solo una entrada.
  3. No contiene ciclos
  4. Si u es el padre y v, v no puede ser padre de u.

  fA : rep −−→ T.D.A. struct nodo .

  Un nodo estar bien formado cuando los campos de hijoizquierdo,padre o hermanosaladerecha son distintos de vacío.

Un nodo raíz se diferencia de otro nodo cuando el padre y el hemanoaladeracha son iguales a 0.

- ***Invariante de la representación:***

  En principio podemos considerar la estructura de árbol de manera intuitiva como una estructura jerárquica. Por tanto, para estructurar un conjunto de elementos ei en árbol, deberemos escoger uno de ellos e1 al que llamaremos raíz del árbol. Del resto de los elementos se selecciona un subconjunto e2,…,ek estableciendo una relación padre-hijo entre la raíz y cada uno de dichos elementos de manera que e1 es llamado el padre de e2,de e3,…ek y cada uno de ellos es llamado un hijo de e1.Iterativamente podemos realizar la misma operación para cada uno de estos elementos asignando a cada uno de ellos un número de 0 o más hijos hasta que no tengamos más elementos que insertar. El único elemento que no tiene padre es e1,la raíz del árbol. Por otro lado hay un conjunto de elementos que no tienen hijos aunque sí padre que son llamados hojas.

En primer lugar definiremos las clases creadas y seguidamente el TDA utilizado en cada una de ellas.

En segundo lugar vamos a describir el funcionamiento del resto de ficheros que componen el programa de QuienEsQuien.

- $QuitarComentarios$: Este fichero es simplemente una función auxiliar que sirve para quitar los comentarios que hay en los ficheros de configuración.
  - El funcionamiento de este archivo se basa en un método en el que cuando en cuenta el carácter # o '\n' omita la lectura y salte a la siguiente línea.

- $Minimo$: Determina si un conjunto de preguntas es minimal, si al eliminar de este conjunto alguna pregunta ya el conjunto de preguntas no es suficiente.
  - En tal caso esa pregunta no es necesaria y por lo tanto el conjunto de partida no es minimal.
  - Con este programa estudiamos las posiciones del código de cada uno de los personajes, si todas ellas tienen el mismo valor, significa que la pregunta que corresponde a esa posición no discriminar a ningún personaje o bandera.

- $Suficiente$: Dado un conjunto de preguntas y un conjunto de personajes con unos rasgos que vienen codificados en un árbol general.
    - Deberemos indicar al usuario si con ese conjunto de personajes y ese conjunto de preguntas se puede determinar de forma unívoca a cada uno de los personajes.
    - Si realizando todas la preguntas para todos los personajes y existiera varias cadenas iguales podríamos decir que no podemos diferenciar unívocamente a los personajes.

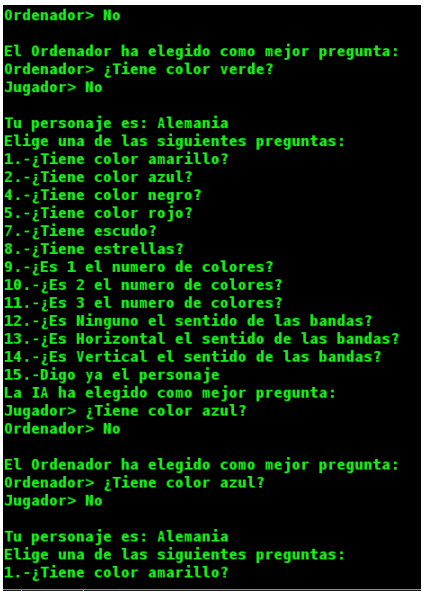
- $Test$: En este código, lee del fichero de configuración que contiene las preguntas posibles, personajes y nombre del fichero con el árbol de rasgos.
    - El programa imprimirá por cada personaje las respuestas a cada una de las preguntas y además dará el código como una secuencia de ceros y unos a esas preguntas.

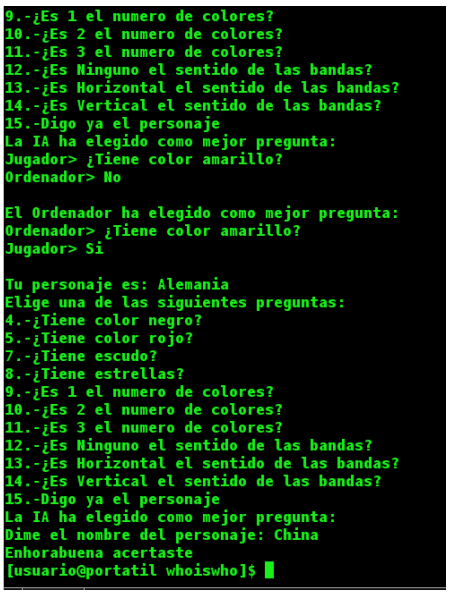
- $Prueba$: Es un programa donde se prueban el resto de funciones que no se usan durante el juego como por ejemplo:
    - Asignar_subarbol
    - Podar_hijomasizquierda
    - Podar_hermanoderecha
    - Insertar_hijomasizquierda
    - Insertar_hermanoderecha
    - Guardar árbol en disco.

- $QuienEsQuien$:En este programa se realiza la lectura de los ficheros de configuración y a continuación se realiza una verificación de jugadores y así decidir correctamente entre ejecutar la IA o pedirle información al jugador por pantalla. Durante la ejecución se la muestra al jugador por pantalla las posibles preguntas que se le puede hacer a la IA y esta responderá según los datos que haya almacenador en el árbol general ,a la vez que efectuara su pregunta y contrastara la respuesta en el mismo árbol general.
    - La IA utiliza un cálculo matemático de probabilidad con el que obtendrá la entropía de la pregunta.
    - comienza el cálculo de la entropia
    - float p_si = count_si/total_personajes, p_no = count_no/total_personajes;
    - float l_si = log2(p_si), l_no = log2(p_no);
    - float suma1 = p_si*l_si, suma2 = p_no*l_no;
    - entropia = -(suma1+suma2);

Programas.

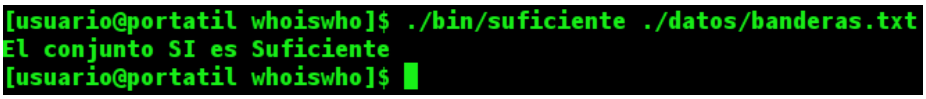
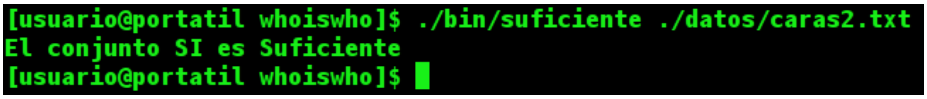
El programa QuienEsQuien tiene varios ficheros para comprobar su funcionamiento entre ellos “Banderas”,”Caras1”y “Caras2”.



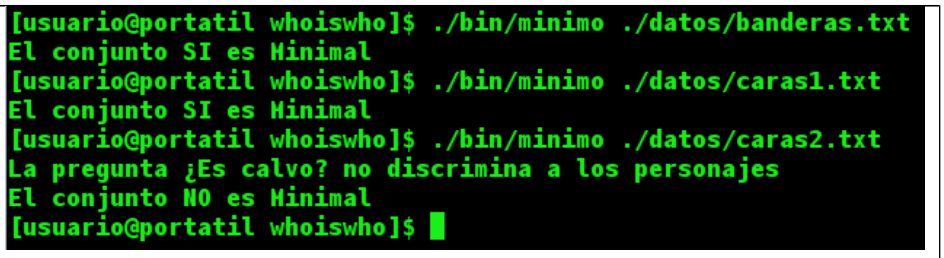


Durante la ejecución de los diferentes programas hemos obtenido un resultado positivo ya que en ocasiones nosotros adivinábamos la bandera o el personaje asignado por el juego de manera aleatoria.

Los resultados obtenidos en cuando a si era **suficiente** con el conjunto de preguntas es el siguiente:IMAGENES

Banderas	
Caras1	
Caras2	

Los resultados obtenidos en cuando a si era **minimal** con el conjunto de preguntas es el siguiente:

Todos	

Diseño y desarrollo de la entropía de la IA

```
const int Jugador::elegirPregunta() {
    float count_si = 0, count_no = 0, num_pregunta = 1;
    float max_entropia = 0.0, entropia = 0.0;
    float total_personajes = this->posibles_personajes.Size();
    Preguntas::iterator it_pregu;
    Personas::iterator it_perso;
    // recorremos las preguntas del jugador
    for (it_pregu = this->posibles_preguntas.begin();it_pregu != this->posibles_preguntas.end(); +
+it_pregu){
        // recorremos los personajes del jugador
        for (it_perso = this->posibles_personajes.begin(); it_perso != this-
>posibles_personajes.end(); ++it_perso){
            // pongo menos 1 por que el codigo empieza en 0 y las preguntas en 1
            if ((*it_perso).GetCodigo()[it_pregu.getNumero()-1] == '1')
                count_si++;
            else
                count_no++;
        }
        entropia = -(((count_no/total_personajes)*log2(count_no/total_personajes))+
((count_si/total_personajes)*log2(count_si/total_personajes)));
        if (entropia>max_entropia){
            max_entropia = entropia;
            num_pregunta = it_pregu.getNumero();
        }else if(this->posibles_personajes.Size() == 1){
            it_pregu = this->posibles_preguntas.end();
            --it_pregu;
            num_pregunta = it_pregu.getNumero() + 1;
        }
        count_si = 0;
        count_no = 0;
    }


    return num_pregunta;
}
```

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 programa es escoger una de las preguntas, que aún no he realizado, de forma aleatoria. Pero existen mejores heurísticas que se propone al alumno 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:

$$H(x)=-\sum p(xi)\log2(p(xi)) \text{ donde}$$

◦ x es la pregunta
◦ xi son las posibles respuesta. En nuestro caso son dos : “Sí” o “No”.
◦ p(xi) es la probabilidad de que se dé la respuesta xi a la pregunta x con un subconjunto del personajes (los no descartados del conjunto de partida).

Para aclarar mejor los conceptos veamos un ejemplo con mayor detalle. Suponiendo el conjunto de personajes de la figura 1 entre las siguiente dos preguntas:
1. ¿Es Mujer su Sexo?
2. ¿Es Claro su Color de Ojos?
Pasamos a calcular la entropía de cada una de las preguntas

$$H(\text{¿EsMujerSuSexo?})=-(p(No)*\log2(p(No))+p(Si)*\log2(p(Si)))$$

donde las probabilidades son: p(No)=10 16 y p(Si)= 6 16 Si hacemos el cálculo obtenemos que

$$H(\text{¿EsMujerSuSexo?})=0.954434$$

Y

$$H(\text{EsClarosuColordeOjos?})=-(p(No)*\log2(p(No))+p(Si)*\log2(p(Si)))$$

donde las probabilidades son:

p(Si)= 9 16 y p(No)= 7 16

Haciendo el cálculo de la entropía obtenemos que
H(¿EsClarosuColordeOjos?)=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.

Como ejemplo de realización mostramos el porcentaje de la entropía para una pregunta en concreto:

```
La IA ha elegido como mejor pregunta:
Este es el valor de la entropia: 0.954434 para la pregunta:¿Tiene color amarillo?
Este es el valor de la entropia: 0.896038 para la pregunta:¿Tiene color azul?
Este es el valor de la entropia: 0.988699 para la pregunta:¿Tiene color blanco?
Este es el valor de la entropia: 0.543564 para la pregunta:¿Tiene color negro?
Este es el valor de la entropia: 0.896038 para la pregunta:¿Tiene color rojo?
Este es el valor de la entropia: 0.896038 para la pregunta:¿Tiene color verde?
Este es el valor de la entropia: 0.988699 para la pregunta:¿Tiene escudo?
Este es el valor de la entropia: 0.811278 para la pregunta:¿Tiene estrellas?
Este es el valor de la entropia: 0.696212 para la pregunta:¿Es 1 el numero de colores?
Este es el valor de la entropia: 0.954434 para la pregunta:¿Es 2 el numero de colores?
Este es el valor de la entropia: 0.988699 para la pregunta:¿Es 3 el numero de colores?
Este es el valor de la entropia: 0.896038 para la pregunta:¿Es Ninguno el sentido de las bandas?
Este es el valor de la entropia: 0.954434 para la pregunta:¿Es Horizontal el sentido de las bandas?
Este es el valor de la entropia: 0.896038 para la pregunta:¿Es Vertical el sentido de las bandas?
Jugador> ¿Tiene color blanco?
Ordenador> Si
```