



INSTITUTO
INGENIERÍA Y
AGRONOMÍA | Carrera
Ingeniería en
Informática


COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS

Trabajo Practico Final

Alumno: Nicolas Garcette

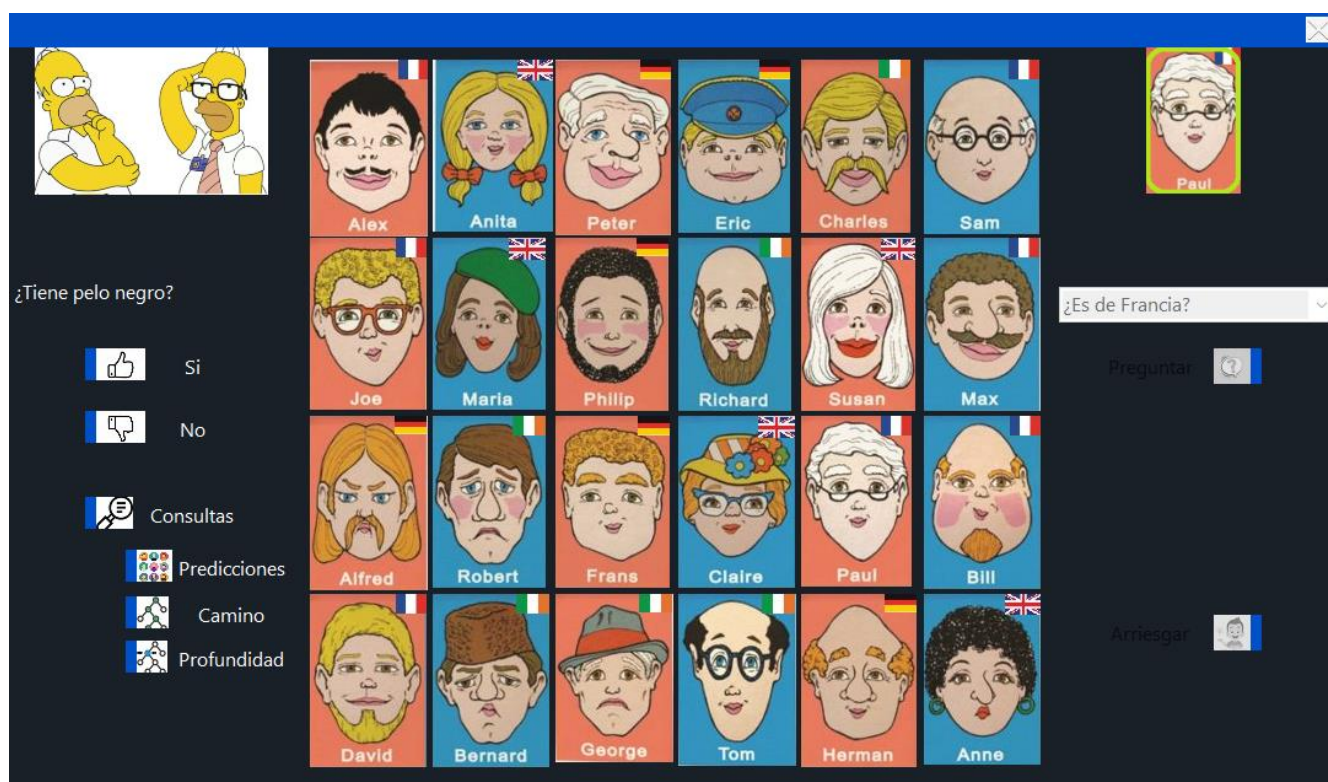
Docente: Leonardo Amet

Comisión: 3

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS		Comisión: 3
			Alumno: Garcette Nicolas
	Trabajo Final		

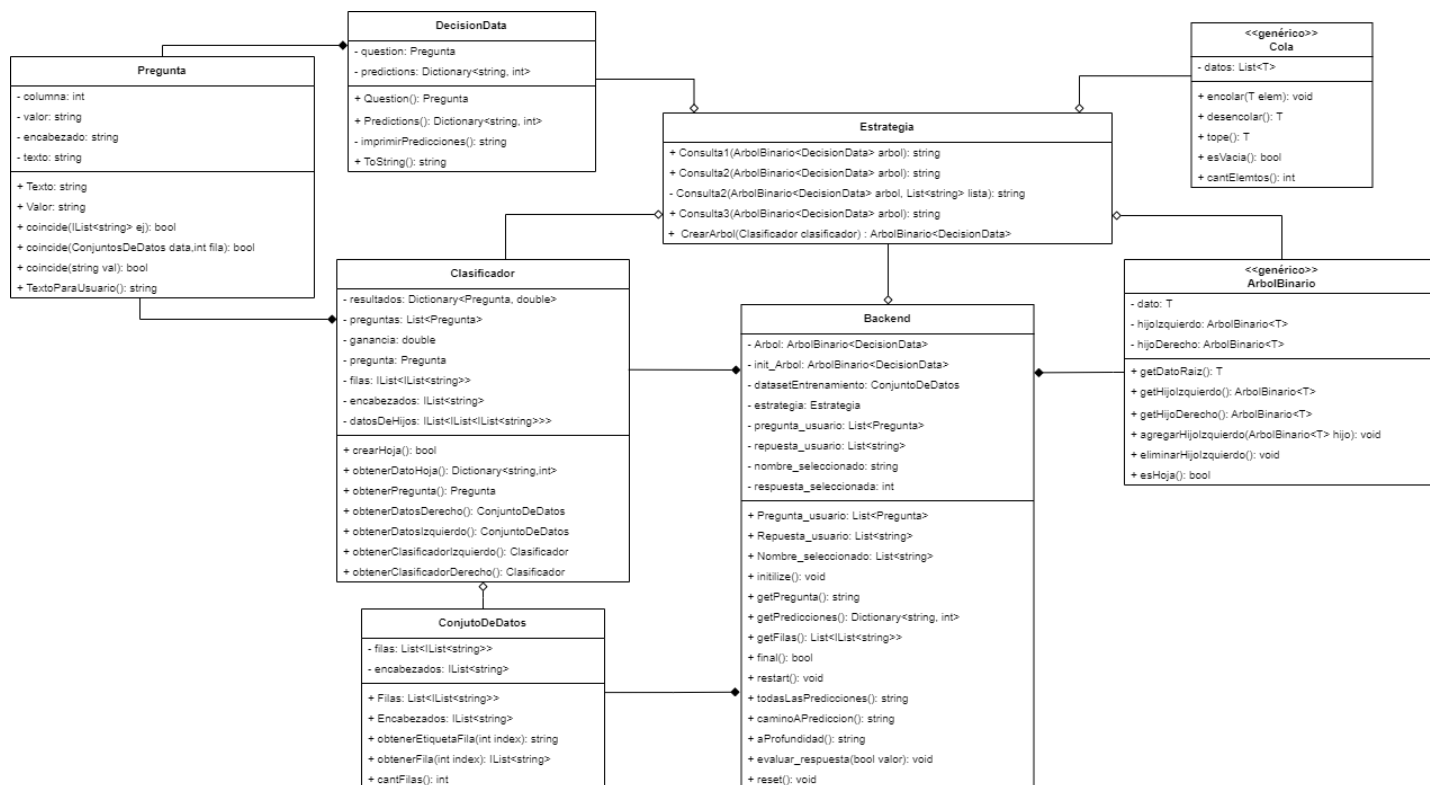
Introducción

En el presente informe se redactara lo trabajado en el trabajo final de la catedra complejidad temporal, estructuras de datos y algoritmos de la carrera ingeniería informática. El cual consiste en codificar y desarrollar la estructura para el funcionamiento del juego, como también realizar métodos de consulta para obtener información del mismo. El juego se llama **“who is who”**, consiste adivinar mediante preguntas el personaje del contrincante (bot), que también deberá descifrar el personaje que ha elegido el jugador. Finalmente, saldrá victorioso el primero que adivine la elección del otro.




-Pantalla principal del juego.

Estructura e implementación del sistema



En la imagen anterior se puede observar que se ha realizado el diseño de clases (UML) describiendo la estructura del sistema. En consecuencia, se obtuvo el entendimiento de las clases, sus atributos, métodos y las relaciones existentes para poder desarrollar el trabajo.

Para la implementación, primeramente el funcionamiento del juego trabaja sobre un árbol de decisión, el cual toma un camino hacia una predicción dependiendo de la elección tomada. Este se desarrolló en un árbol binario compuesto de nodos-decisión, que vendrían a ser las preguntas para adivinar el personaje, y los nodos-hoja que serían el resultado o predicción de esa concatenación de preguntas. Para su desarrollo se utilizó la clase **Estrategia**, la cual contiene los siguientes métodos: **CrearArbol ()**, **Consulta1 ()**, **Consulta2 ()**, **Consulta3 ()**. Todas serán explicadas a continuación.

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	

Crear árbol

```

public ArbolBinario<DecisionData> CrearArbol(Clasificador clasificador)
{
    ArbolBinario <DecisionData> arbol;

    si es nodo-hoja se obtiene la prediccion y se retorna el nodo.
    if (clasificador.crearHoja()) {
        // con la prediccion se crea el nodo-hoja.
        arbol = new ArbolBinario<DecisionData>(new DecisionData(clasificador.obtenerDatoHoja()));
        return arbol;
    } else {
        se obtiene la pregunta y se convierte en nodo-desicion
        arbol = new ArbolBinario<DecisionData>(new DecisionData(clasificador.obtenerPregunta()));

        se obtiene el ClasificadorIzquierdo
        Clasificador izquierdo = clasificador.obtenerClasificadorIzquierdo();
        se llama a crearArbol para que retorne el arbol del nodo izq
        ArbolBinario<DecisionData> hijoIzq = this.CrearArbol(izquierdo);

        se obtiene el ClasificadorDerecho
        Clasificador derecho = clasificador.obtenerClasificadorDerecho();
        se llama a crearArbol para que retorne el arbol del nodo izq
        ArbolBinario<DecisionData> hijoDer = this.CrearArbol(derecho);

        se agregan los hijos a su nodoPadre.
        arbol.agregarHijoDerecho(hijoDer);
        arbol.agregarHijoIzquierdo(hijoIzq);


        return arbol;
    }
}

```

EL método **crear árbol** es el que construye la estructura de árbol de decisión para que el bot pueda descifrar el personaje elegido por el jugador. Este método es un algoritmo recursivo y toma como parámetro un **Clasificador**, que es el que realiza una propiedad de orden adecuada de las preguntas y predicciones. Finalmente, retorna un árbol Binario de tipo Decisión data, hay que recalcar que esta última clase toma como único parámetro una pregunta o una predicción.

Analizando el algoritmo, se observa que el **caso base** consulta si el clasificador es hoja, en caso afirmativo se instancia un árbol binario pasándole como parámetro un Decisión data, asimismo como el clasificador era hoja deducimos que contiene una predicción, la cual obtenemos mediante **ObtenerDatoHoja**, pasándosela como parámetro a decisión data.

En caso contrario, como el clasificador no es hoja contiene una pregunta, por lo cual se obtiene la pregunta mediante **ObtenerPregunta** y se instancia un árbol binario de tipo decisión data teniendo como parámetro la pregunta. Seguido, se obtiene el clasificador derecho y el izquierdo, para obtener recursivamente los subárboles de dichos clasificadores, para ello se llama a **CrearArbol (izquierdo)** y **CrearArbol**

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	

(derecho), y se almacena el retorno en **hijoDer** e **hijoIzq** los cuales luego se agregan al árbol. Cabe aclarar que como es un algoritmo recursivo la creación del árbol se dará desde las hojas hacia los nodos internos, por lo cual se conectaran al respectivo padre una vez esté formado el subárbol.

Cabe aclarar que podría hacerse todo en menos líneas de código, ya que se hacen algunas instancias demás, que podrían ahorrarse asignaciones pero se optó en dejarlo por partes para una mayor comprensión. Después de todo, es el método que más trajo problemas a implementar ya que en un principio al desconocer cómo funcionaban y se relacionaban las clases se complicó la creación del árbol, pero a medida que se indagaba en el código y varios procesos de prueba, y error se creó el árbol.

Consulta 1


```
public String Consulta1(ArbolBinario<DecisionData> arbol)
{
    string resutl = "";
    si es hoja, retorna la prediccion.
    if (arbol.esHoja()) {
        return arbol.getDatoRaiz() + "\n";
    }

    si no es hoja, se busca en los hijos
    if (arbol.getHijoIzquierdo() != null) {
        resutl += Consulta1(arbol.getHijoIzquierdo()); // consulta al hijoIzquierdo.
    }
    if (arbol.getHijoDerecho() != null) {
        resutl += Consulta1(arbol.getHijoDerecho()); // consulta al hijoDerecho.
    }

    return resutl;
}
```

La funcionalidad del algoritmo consulta1 es retornar en un string el cual luego se mostrara en pantalla al jugador. Este string contiene todas las posibles predicciones posibles al momento de la consulta.

El método se podría decir que es una adaptación de recorrido pre-orden en donde se procesa la raíz con un caso base y recorre el árbol desde los hijos izquierdos a los derechos. Para su funcionalidad se requiere encontrar una hoja, que es en donde se almacenan las predicciones. Por ese motivo, como caso base se consulta si el nodo es hoja utilizando el

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	

método **esHoja ()** el cual devuelve un bool. En caso de que el bool sea TRUE, retorna el dato del nodo el cual se va almacenar en su respectiva llamada.


Por otro lado en caso de que el nodo no sea hoja, se va por los hijos en donde en caso de que no sean nulos se llama recursivamente **Consulta1**, y el retorno se almacena en "resultado" el cual es un string declarado previamente, para luego ser retornado al final del método.

Consultas

```

Richard:100%
Charles:100%
Max:100%
Alfred:100%
Anita:100%
Peter:100%
Robert:100%
Tom:100%
Sam:100%
Joe:100%
Paul:100%
Claire:100%
Susan:100%
Frans:100%
Herman:100%
Bill:100%
David:100%
Eric:100%
Maria:100%
Bernard:100%
George:100%
Philip:100%
Alex:100%
Anne:100%
```

-Predicciones de la consulta 1.

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	

Consulta 2

```

public String Consulta2(ArbolBinario<DecisionData> arbol){
    List<string> lista = new List<string>(); //se agrega un list para que el camino almacenado persista.
    return Consulta2(arbol,lista);
}
retorna todos los caminos hacia la prediccion.
private String Consulta2(ArbolBinario<DecisionData> arbol, List<string> lista){

    string resutl = "";

    lista.Add(arbol.getDatoRaiz().ToString()); // se guarda el dato
    si es hoja, devuelve el camino almacenado
    if (arbol.esHoja()) {
        foreach (var ele in lista) {
            resutl += ele + " | ";
        }
        return resutl;
    }

    if (arbol.getHijoIzquierdo() != null) {
        resutl += Consulta2(arbol.getHijoIzquierdo(), lista) + "\n";
    }
    lista.RemoveAt(lista.Count - 1);
    if (arbol.getHijoDerecho() != null) {
        resutl += Consulta2(arbol.getHijoDerecho(), lista) + "\n";
    }

    return resutl;
}


```

Se ha desarrollado el método consulta2, el cual es el encargado de retornar todos los caminos hacia las hojas, eso equivale a todas las preguntas que se deben hacer para llegar hacia una predicción.

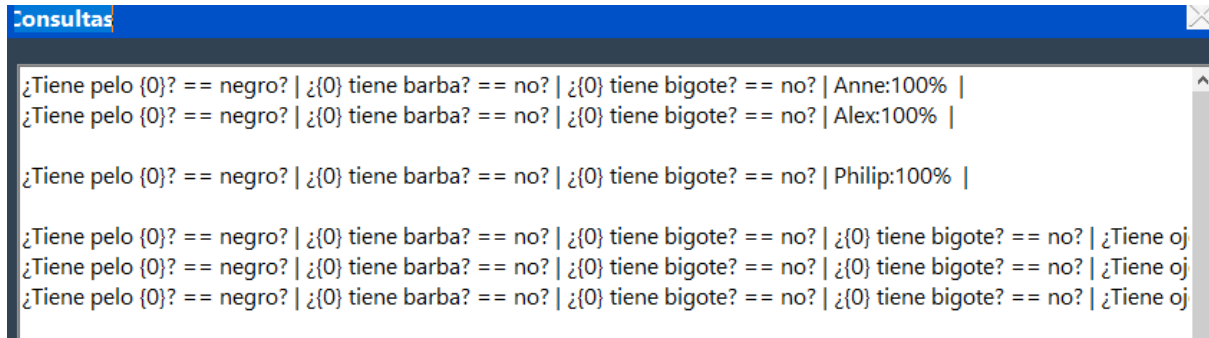
Para resolver esta consulta, se ha creado un método público **Consulta2** que recibe como parámetro el árbol y contiene una lista, la cual se utilizara para almacenar los diferentes caminos que se vayan acumulando. Además, es la encargada de llamar al método privado **Consulta2** pasándole como parámetro la lista y el árbol.

El método privado es en donde se obtiene el resultado para cumplir con la funcionalidad que se ha mencionado. También se implementó como una adaptación de recorrido pre orden, en el cual se comienza agregando el dato del nodo en la lista, seguido de esto tenemos el caso base, donde se consulta con un **if** si el nodo es hoja, en caso de serlo significa que se encontró una predicción por lo cual con un **foreach** se correrá la lista y se acumula todo el camino hasta la predicción en un string, luego el mismo se retorna.

Seguido de esto se llama recursivamente a consulta2 pasándole el hijo izquierdo y la lista, retorno se acumula en un string. Luego, se quita el último elemento de la lista ya que

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	

corresponde al hijo izquierdo siendo este un camino distinto. Finalmente, se hace el proceso recursivo con el hijo derecho.



-Consulta 2 realizada.

Consulta3

```

public String Consulta3(ArbolBinario<DecisionData> arbol){

    Cola<ArbolBinario<DecisionData>> c = new Cola<ArbolBinario<DecisionData>>();
    ArbolBinario<DecisionData> aux ;
    int contNiv = 0;
    c.encolar(arbol); //se encola el arbol
    c.encolar(null); //se encola el separador para cuantificar el lvl.
    string result = "\nNivel: "+contNiv+"\n";

    while (!c.esVacia()) {
        aux = c.desencolar();


        if (aux == null) {
            contNiv++;
            if (!c.esVacia()){
                c.encolar(null); //si aun quedan elementos, encola separador.
                result += "\nNivel: "+contNiv+"\n";
            }
        } else {
            result += aux.getDatoRaiz().ToString();

            if (aux.getHijoDerecho() != null)
                c.encolar(aux.getHijoDerecho());

            if (aux.getHijoIzquierdo() != null)
                c.encolar(aux.getHijoIzquierdo());
        }
    }
    return result;
}

```

La consulta 3 cumple la función de retornar los datos de los nodos diferenciados por niveles. En la implementación se optó por un recorrido por niveles con separador para identificar

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS	Comisión: 3
		Alumno: Garcette Nicolas
	Trabajo Final	


cada nivel. En primer lugar se crea una clase Cola, donde lo primero que se almacena es el árbol que se pasa como parámetro, seguido se encola un null como separador de nivel, además se utiliza un contNiv para contar los niveles. Acto seguido entrara en bucle hasta que la cola no se vacié. Una vez en bucle se irán desencolando los arboles almacenándolos en un árbol auxiliar, el cual se verifica si es un null, en caso de no serlo se procesa el auxiliar almacenando su dato en un string que acumulara todos los resultados. Luego, se encolaran sus hijos tanto el izquierdo como el derecho. Por otro lado, en caso de que se desencole un null, significa que los hijos seguidos del null se encuentran en otro nivel por lo cual se incrementa el contador de niveles, además, se agrega al acumulador el nivel en que se encuentra y si la cola no está vacía se vuelve a encolar null, esto se realiza para que no entre en un bucle infinito de nulls.

```
Nivel: 0
¿Tiene pelo {0}? == negro?
Nivel: 1
¿{0} tiene bigote? == no? ¿{0} tiene barba? == no?
Nivel: 2
¿Tiene ojos {0}? == azules? ¿Tiene ojos {0}? == marrones? Philip:100% ¿{0} tiene bigote? == no?
Nivel: 3
¿{0} tiene barba? == no? Alfred:100% ¿{0} tiene gafas? == si? ¿{0} tiene gafas? == no? Alex:100% Anne:100%
Nivel: 4
Richard:100% ¿Tiene pelo {0}? == marrones? ¿Tiene pelo {0}? == marrones? Tom:100% ¿{0} tiene sombrero? == si? ¿{0} ti
Nivel: 5
Charles:100% Max:100% ¿Tiene pelo {0}? == blanco? Robert:100% ¿Tiene pelo {0}? == blanco? Claire:100% ¿{0} tiene bar
Nivel: 6
Anita:100% Peter:100% ¿Tiene pelo {0}? == rubio? Paul:100% ¿Es de {0}? == Alemania? ¿Tiene pelo {0}? == rubio? ¿Tiene
Nivel: 7
Sam:100% Joe:100% Susan:100% ¿Tiene pelo {0}? == calvo? Bill:100% David:100% Eric:100% ¿Es de {0}? == Irlanda?
Nivel: 8
Frans:100% Herman:100% Maria:100% Bernard:100%
```

-Consulta 3 diferenciada por niveles.

Sugerencia para una versión mejorada del mismo.

Se mencionaran algunas de las ideas o sugerencias que podrían realizarle al sistema para una versión avanzada del mismo. En primer lugar, estaría bueno desarrollar un apartado de las mismas consultas desarrolladas pero para que le sean de ayuda al jugador, de modo que un árbol de decisión tome las respuestas del bot, ya que en el juego las consultas son en base a las respuestas del jugador. Siguiendo con la lógica de ayudar al jugador, sería

	COMPLEJIDAD TEMPORAL, ESTRUCTURAS DE DATOS Y ALGORITMOS		Comisión: 3
			Alumno: Garcette Nicolas
	Trabajo Final		

mucho más atractivo a medida que se descartan predicciones se tachan dinámicamente los personajes, no obstante, debe ser más complejo de realizar.

Por otro lado, otra de las sugerencias sería que cuando finaliza el programa en vez de tener la opción de inicio, sea una opción de **‘volver a jugar’** la cual te regrese a elegir personaje, para una mejor experiencia de usuario. Consecuentemente, se analizó realizar esta pequeña ‘mejora’, la cual se logró modificando el método **button7_Click** de la clase **Form2**, **en vez de** instanciar **inicio** y mostrarlo, se elimina esa instancia y se realizó una instancia de **formUser** la cual toma como parámetro un **form1** y un **true** para poder iniciar, seguido de esto se muestra mediante un **.Show**. Además, se cambió el nombre del botón a **‘jugar’**.



-botón modificado a **‘jugar’**.

Conclusión

Con respecto al trabajo final de CTEDyA, fue de gran ayuda para reforzar lo aprendido en la cátedra, en ellos se destaca la implementación de árboles binarios y algunos de sus recorridos, pero tanto la práctica que se realizó como también el razonamiento que se generó para comprender y afrontar el trabajo, fue muy favorable para afianzar los conocimientos adquiridos durante la cursada.