

**Trabajo Final**

**Complejidad Temporal, Estructuras de Datos y Algoritmos**

**Carrera: Ingeniera en Informática.**

**Alumno: Augusto Di Bona.**

**Profesor: Leonardo Amet.**

**Comisión: 2.**

**Introducción**

Este trabajo consiste en hacer un videojuego con una IA como contrincante. Este juego va a tener 12 cartas (6 cartas para la IA, 6 cartas para el Humano) las cuales se repartirán aleatoriamente a cada jugador. El jugador que supere el límite del montículo será el perdedor. Cada jugador sabe las cartas que tiene el contrincante y el límite actual del montículo.

La IA entre sus posibles cartas deberá escoger la mejor jugada para poder vencer al Humano, es decir, la IA siempre va a jugar a ganar, por esta razón la IA debe tener cierta inteligencia.

Para lograr que la IA sepa cada jugada y que pueda reconocer cual es la mejor jugada, debemos hacer dos pasos:

1) Utilizar un Árbol General, en donde en cada raíz se almacenará una jugada (carta), es decir, debemos de construir un árbol con todas las posibles jugadas (cartas) que pueda haber en el juego, por esta razón lo llamaremos **Árbol de Jugadas**. Con este árbol la IA podrá saber todas las posibles jugadas en el juego.

2) Crear un algoritmo llamado **minMax**, la cual consiste en la elección del mejor movimiento para el computador. Una vez que tenemos nuestro **Árbol de Jugadas**, con **minMax** vamos a hacer que cada jugada (carta) tenga un valor que indique si la IA puede ganar o puede perder. De esta forma, la IA va analizar sus cartas y a seleccionar el mejor movimiento para poder vencer al Humano.

Nota: Dicho valor que indica si la IA gana o pierde, lo llamaremos **valor heurístico** o **función heurística**. Entonces, podríamos llamar a este árbol como, **Árbol de Jugadas Heurístico**.

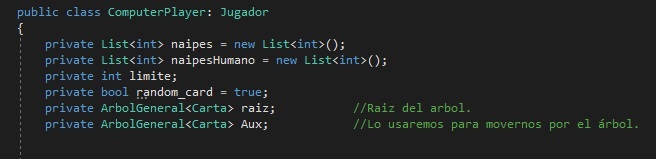
**Imágenes de las pantallas que componen el sistema codificado.**

**Nota: Algunas de las imágenes ya tiene comentarios que explican la funcionalidad.**

**Clase ComputerPlayer.**

La clase ComputerPlayer será la que tenga internamente el árbol de jugadas del juego.

**Variables.**

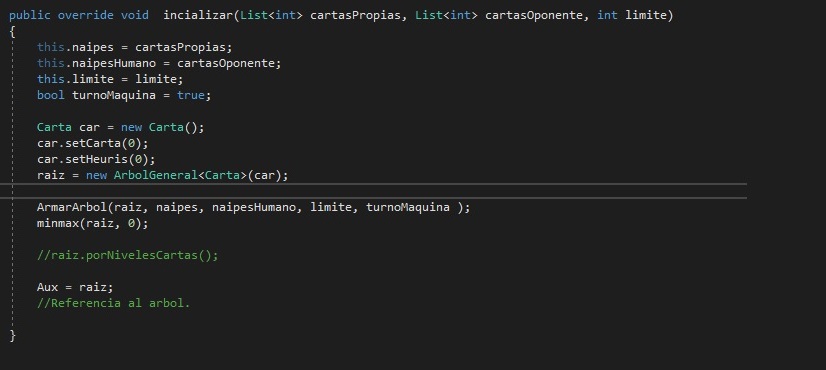
****

Las variables que tendremos en ComputerPlayer serán la lista de naipes de la IA y del Humano, un límite de cartas, un árbol general llamado “raíz” la cual almacenará datos de tipo “Carta”. Dicho árbol será nuestro **árbol de jugadas**.

Por último, tendremos una variable “Aux” de tipo ArbolGeneral la cual va a hacer referencia al punto actual en el que se encuentra el juego. Dicha variable la utilizaremos para hacer consultas durante el juego.

**Inicializar.**

Esta función lo que hará será inicializar las cartas de la IA, del humano, y el límite. También lo que hará este método es hacer una llamada a la función **ArmarArbol** la cual va a crear el árbol de jugadas, y hará una llamada a la función **minMax** para establecer los valores heurísticos de cada jugada.

****

El método **inicializar** recibirá por parámetro una lista de cartas propias, una lista de cartas del Oponente, y un límite en el juego.

Inicializaremos las cartas de la IA, del humano, y el límite.

Tendrá una variable local “turnoMaquina” en “true”.

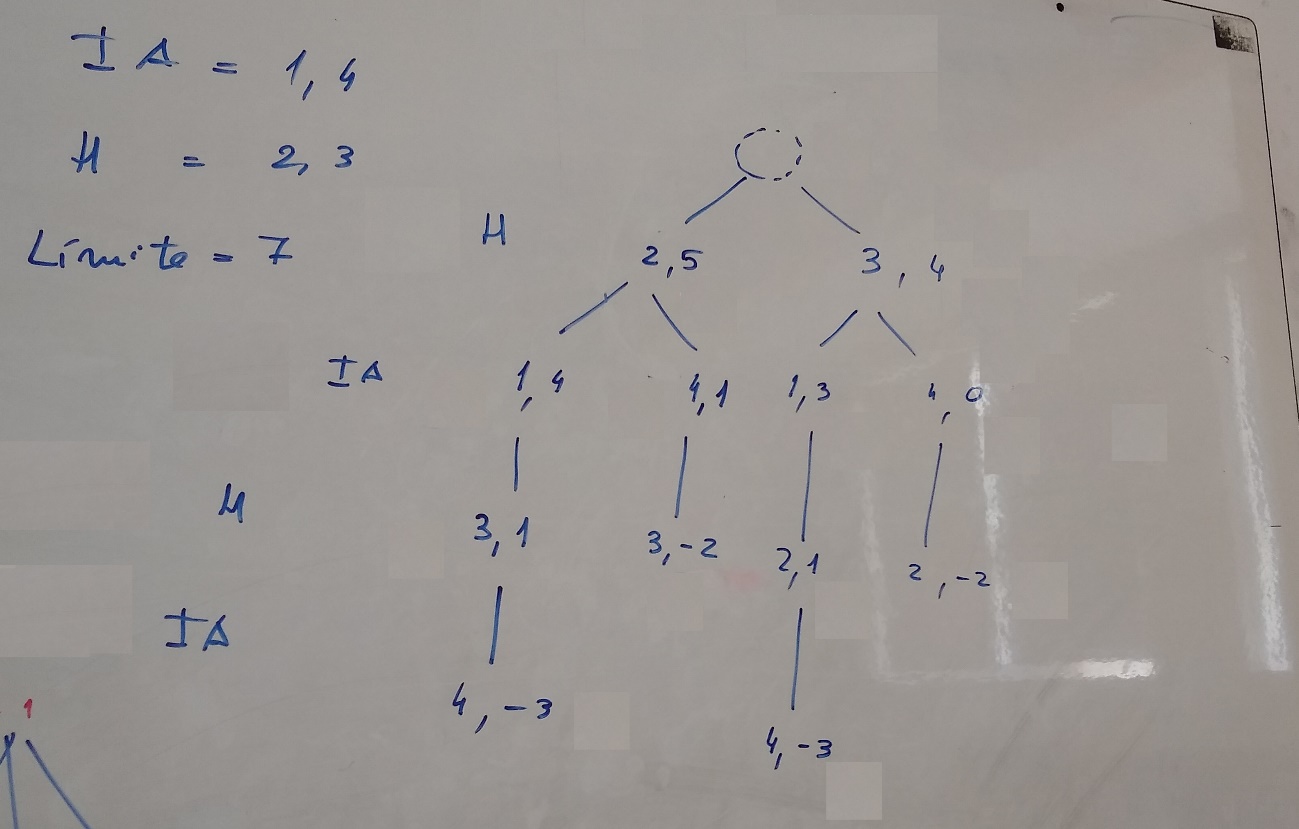
Crearemos un objeto de tipo Carta llamado “car” con un valor de carta “0” y un valor heurístico de “0”.

Vamos a crear un objeto de tipo ArbolGeneral la cual tendrá en su raíz el objeto “car” que hemos creado, e inicializaremos la variable “raíz”.

Luego, se ejecutarán los métodos **ArmarArbol** la cual creara los hijos de nuestro árbol y formara las **jugadas.**, y el método **MinMax** la cual asignara los valores heurísticos de cada jugada del árbol.

**ArmarArbol**

Esta función lo que hará será crear el **árbol de jugadas**. Para entenderlo veamos como funciona con pocas cartas, por ejemplo: 4 cartas, y un límite de 7.



Como vemos, la raíz no posee ningún valor, y esto se debe a que no va a representar ninguna jugada del juego. Los próximos niveles del árbol son los que representaran las jugadas del Humano y la IA.

El humano siempre tiene el 1er turno, por lo tanto, las posibles jugadas del humano están representadas en el nivel 1 del árbol. El humano tiene las cartas (2 y 3), por lo tanto, el nivel 1 tendrá las posibles cartas que el Humano puede jugar.

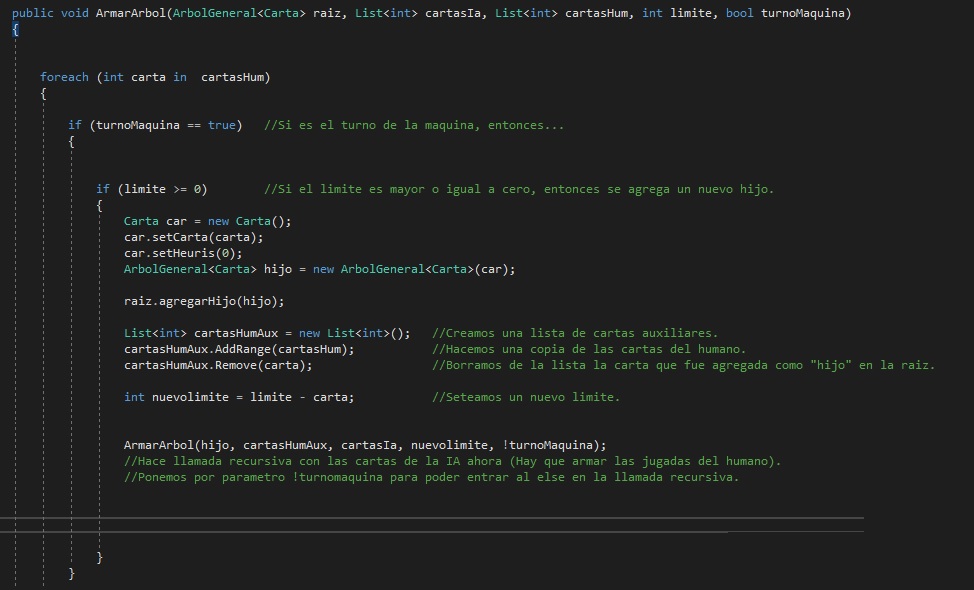
La IA siempre tiene el 2do turno, por lo tanto, las posibles jugadas de la IA están representadas en el nivel 2 del árbol. La IA tiene las cartas (1 y 4), por lo tanto, el nivel 2 tendrá las posibles cartas que la IA puede jugar.

Nota: En cada carta estamos especificando el límite actual, es decir, si el humano juega la carta “2” entonces el límite valdrá 5 (7-2=5), o si el humano juega la carta “3” entonces el límite valdrá 4 (7-3=4). La información del límite no debe estar almacenada en cada nodo del árbol, esto es solo para entender lo que está sucediendo. (Mas adelante veremos cómo y en donde actualizamos el límite).

En los siguientes niveles debemos tener cuidado con las jugadas que representamos en el árbol, ya que si por ejemplo, el Humano juega la carta “2” en el nivel 1, entonces dicha carta ya no existe en la lista de cartas del Humano, por lo tanto, la única posible carta que el humano puede jugar en el próximo turno es la carta “3”. Otra posibilidad es que, si el Humano juega la carta “3” en el nivel 1, entonces dicha carta ya no existe en la lista de cartas del Humano, por lo tanto, la única posible carta que el humano puede jugar en el próximo turno es la carta “2”.

Una vez que el limite de la jugada sea negativo, quiere decir que dicho jugador perdió, por lo tanto, el juego termina en ese punto y el nodo no tendrá hijos. Por ejemplo, si el humano juega la carta “2”, luego la IA juega la carta “1”, luego el humano juega la carta “3”, y luego la IA juega la carta”4”, el limite será -3, por lo tanto, la IA perdió el juego.

**ArmarArbol** - **Implementación**

****

Este método recibe por parámetro la raíz del árbol, la lista de cartas de la IA y el Humano, un limite de cartas, y una variable booleana que representa el turno de la IA. Básicamente con este método armaremos un árbol con todas las jugadas del juego de manera **recursiva**.

Primero se va a hacer un **foreach** de las cartas del Humano, (más adelante explicaremos lo que haremos con cada carta.) ¿Por qué empezamos haciendo un **foreach** con las cartas del Humano? Porque el humano siempre tiene el primer turno en el juego, por lo tanto, el primer nivel del árbol tenemos que armarlo primero con las cartas del Humano. El nivel 0 no representa ningún turno, pero en la lógica del código la raíz será el turno de la IA.

Luego tenemos un **if** con la condición de si “turnoMaquina es igual a true”. TurnoMaquina recordemos que se inicializa en true, por lo tanto, al inicio del juego entrara primero a este **if**. (El else lo analizaremos más adelante.)

Ahora tenemos un **if** con la condición de que el limite sea mayor o igual a 0, donde si esta condición es verdadera, entonces se agrega un nuevo hijo a la raíz actual. Supongamos que el limite al iniciarse el juego sea “30”, entonces se va a cumplir la condición y entrara al **if**. Entonces, cuando entremos al **if**, lo primero que haremos será crear un nuevo objeto de tipo Carta y le setearemos su valor de carta pasándole por parámetro la carta obtenida del foreach. El valor heurístico de la Carta lo seteamos en “0”, (de asignarle un valor “-1” o “+1” se encargará más adelante la función **minMax**).

Vamos a crear un nuevo ArbolGeneral llamado “hijo” y le pasaremos por parámetro la Carta que hemos creado. Dicho “hijo” lo agregaremos como un nuevo hijo de la raíz actual del árbol. Con esto ya logramos agregar una jugada del Humano en el primer nivel del árbol.

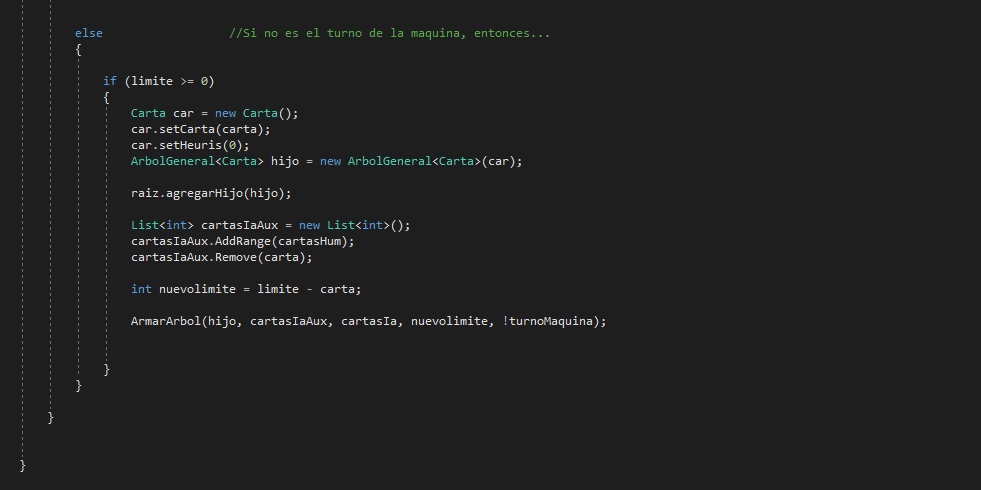
Lo siguiente será crear una lista llamada “CartasHumAux” la cual tendrá una copia de la lista de cartas del Humano, y a dicha lista auxiliar le eliminaremos la carta obtenida del foreach, ya que dicha carta ya la agregamos en el árbol de jugadas. Esta lista Aux lo que hace entonces es representar las cartas del Humano que aún nos quedan agregar en el árbol de jugadas.

Ahora que hemos agregado una carta al árbol debemos disminuir el límite, por lo tanto, tendremos una variable llamada “nuevoLímite” que será igual al limite menos el valor de la carta obtenida del foreach.

Por último, haremos una llamada recursiva de **ArmarArbol** y le pasaremos por parámetro el “hijo” de tipo ArbolGeneral que creamos, las cartas “CartasHumAux” que contienen las cartas del Humano que aún nos quedan agregar, las cartas de la IA, el “nuevoLimite”, y el “turnoMaquina” en false.

Notas:

* El “turnoMaquina” lo ponemos en false ya que al meternos en la recursión estaremos parados sobre el nivel del Humano, y no queremos que vuelva a entrar al **if** con la condición de si “turnoMaquina es igual a true”.
* El orden de los valores por parámetro en la llamada recursiva **ArmarArbol** es diferente, es decir, la función **ArmarArbol** recibe por parámetro los valores en el siguiente orden: (Raiz, CartasIA, CartasHum, limite, turnoMaquina), mientras que en la llamada recursiva de **ArmarArbol** recibe por parámetro los valores en el siguiente orden: (Raiz, CartasHumAux, CartasIA, nuevoLimite, turnoMaquina). Como vemos cambiamos de lugar las cartas del humano con las cartas de la IA. Esto se debe a que al meternos en la recursión estaremos parados sobre el nivel del Humano (nivel 1), y cuando entremos al foreach vamos a querer que recorra las cartas de la IA para poder armar las posibles jugadas de la IA (nivel 2) en el árbol de jugadas.

Ahora analicemos el else.

En el else estará la condición si turnoMaquina es igual a false, por lo tanto, el else representa el turno del Humano. El código es igual al explicado anteriormente en el **if**. La diferencia es que en la llamada recursiva **ArmarArbol** vamos a volver a cambiar el orden de los valores por parámetro, ahora **ArmarArbol** recibe por parámetro los valores en el siguiente orden: (Raiz, CartasIaAux, cartasIa, nuevoLimite, turnoMaquina). Esto se debe a que al meternos en la recursión estaremos parados sobre el nivel de la IA (nivel 2), y cuando entremos al foreach vamos a querer que recorra las cartas del Humano para poder armar las posibles jugadas del Humano (nivel 3) en el árbol de jugadas.

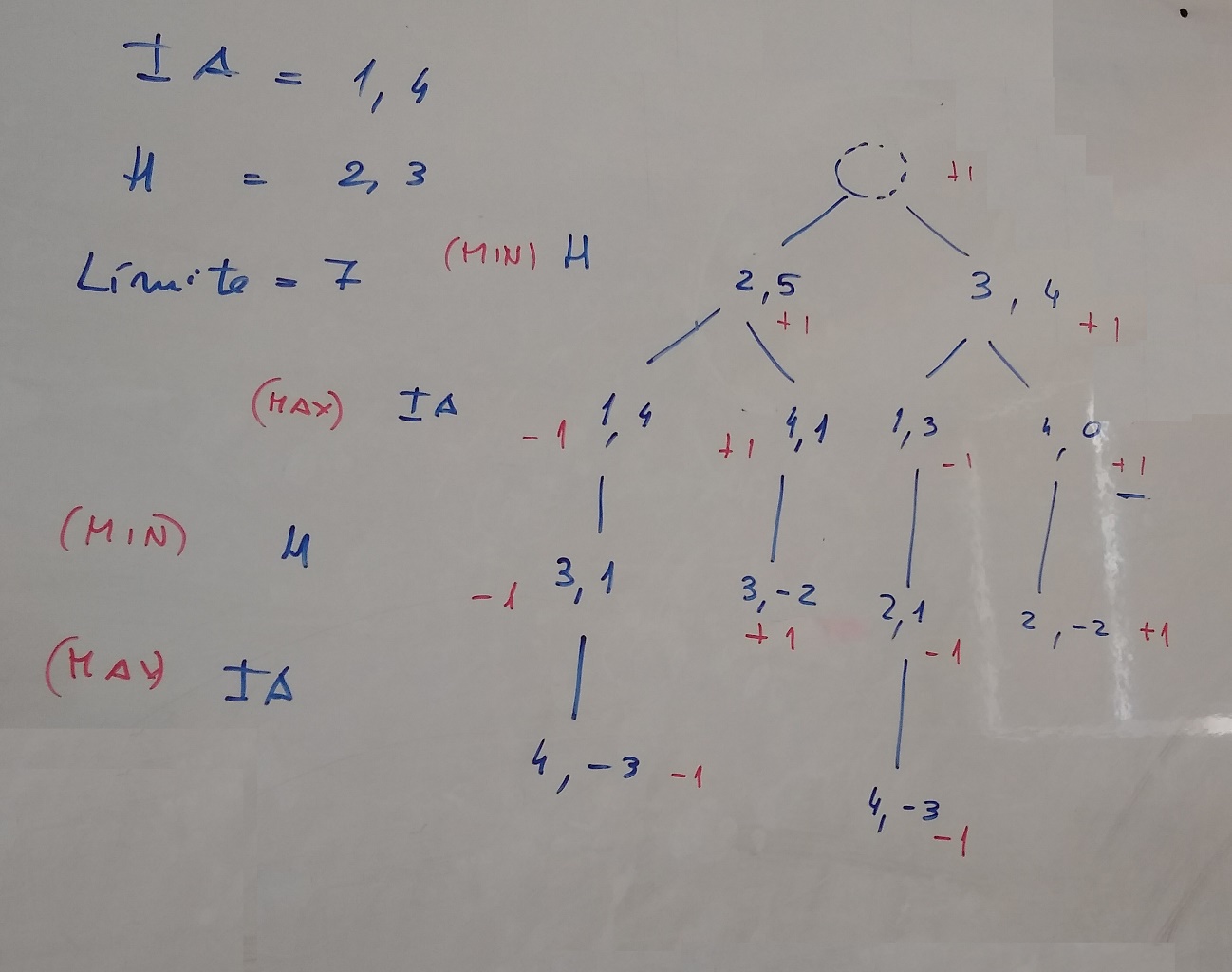
Nota:

* A **ArmarArbol** le pasamos por parámetro (Raiz, CartasIaAux, cartasIa, nuevoLimite, turnoMaquina), donde la variable “cartasIa” tiene las cartas del Humano.

**minMax**

El algoritmo **MinMax** consiste en una **función Heurística**, la cual es un valor que se la asigna a cada jugada. Nota: La función Heurística no es una función propiamente dicha, sino que es un valor que se le asigna a cada jugada y lleva el nombre de **función Heurística**.

Dicho valor sirve para orientarse en la toma de decisiones de la IA, es decir, si una jugada tiene el valor heurístico “+1” quiere decir que la IA va a ganar el juego si utiliza dicha carta, pero si una jugada tiene el valor heurístico “-1” quiere decir que la IA va a perder el juego si utiliza dicha carta. Para entenderlo veamos cómo funciona con el ejemplo anterior de 4 cartas, y un límite de 7.



Como dijimos, la función **MinMax** consiste en establecer los valores heurísticos (+1 o -1) en cada jugada. Si la jugada tiene un valor “+1” quiere decir que la IA va a ganar en dicha jugada, y si la jugada tiene un valor “-1” quiere decir que el Humano va a ganar en dicha jugada.

El algoritmo es recursivo y se centra el llegar a las hojas del árbol, una vez encontrada una hoja va a comprobar si dicha hoja se encuentra en el nivel de la IA o del Humano. Si es el nivel del humano quiere decir que fue su turno y perdió, por lo tanto, en dicha hoja vamos a setear que tiene un valor heurístico de “+1”, pero si es el nivel de la IA quiere decir que fue su turno y perdió, por lo tanto, en dicha hoja vamos a setear que tiene un valor heurístico de “-1”,

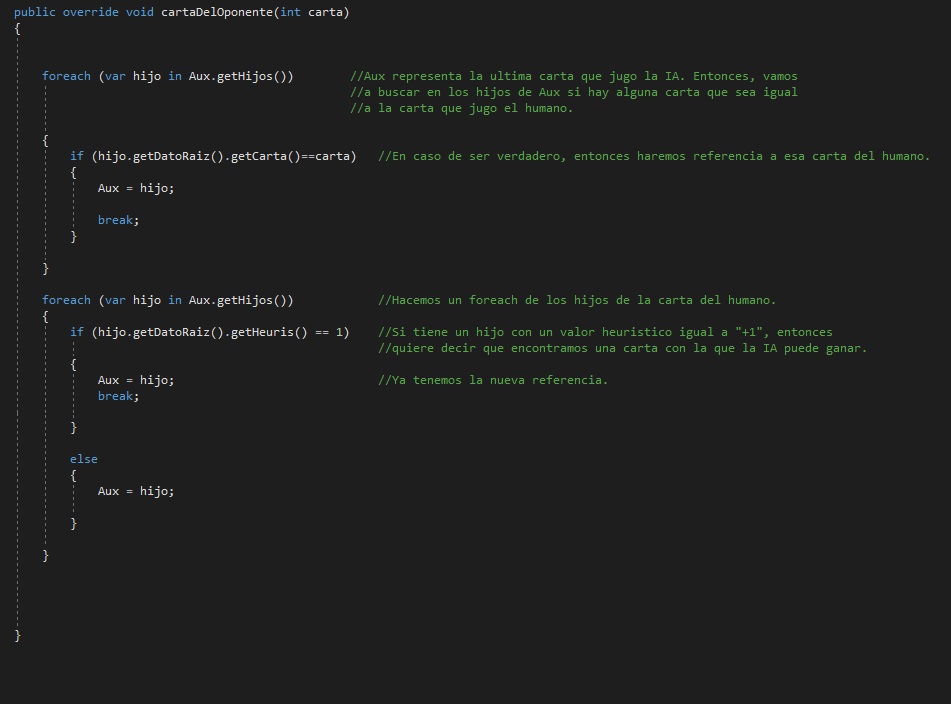
Ahora solo queda ir volviendo de la recursión, e iremos seteando los valores heurísticos de cada nodo en base a sus hijos. Por ejemplo, la IA tiene una hoja donde juega la **carta 4** y pierde, por lo tanto, la **carta 4** tiene un valor heurístico de “-1”. Ahora volvemos un paso atrás en la recursión y estamos parados en la **carta 3** del Humano. Dicha carta va a pedirle a sus hijos cuales sos sus valores heurísticos, en este caso la **carta 3** tiene solo un hijo (**carta 4**), por lo tanto, la **carta 3** obtendrá el valor heurístico “-1” y se lo seteara así mismo. Ahora la **carta 3** tiene el valor heurístico “-1”.

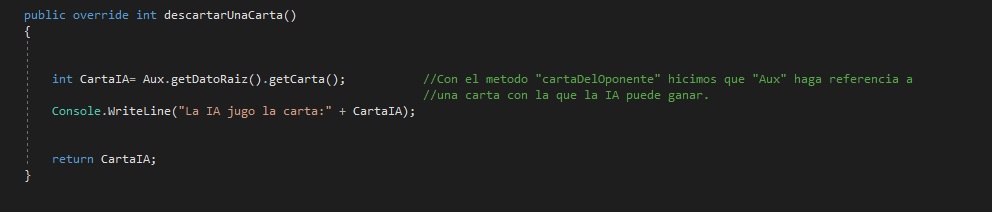
Supongamos otro caso donde volvamos un paso atrás en la recursión y dicho nodo tenga más de un hijo, por ejemplo, la **carta 2** del Humano en el nivel 1, ¿Con cuál valor heurístico va a quedarse la **carta 2**? ¿Con “-1” de la **carta 1**, o “+1” de la **carta 4**? Se quedará con el valor “+1” y se lo seteara así mismo. Ahora la **carta 2** tiene el valor heurístico “+1”. ¿Por qué se quedo con +1? Porque la IA siempre juega a ganar, siempre juega la jugada con el valor heurístico max, o sea, si el humano juega la **carta 2** en su primer turno, entonces la IA va a jugar la **carta 4** en su turno haciendo que pierda el Humano, la IA nunca jugaría la **carta 1** ya que con esa jugada perdería. Entonces, la **carta 2** del Humano va a tener un valor heurístico de “+1” indicando que va a perder si juega dicha carta. Este mismo razonamiento se aplica con el Humano, es decir, el Humano siempre intenta jugar a ganar, siempre juega la jugada con el valor heurístico min.

Nota: La IA cuando esté jugando, lo que hará será buscar estos valores heurísticos de “+1”, ya que esas son las jugadas que lo harán ganar y que el Humano pierda.

Nota: La **implementación** de **minMax** se encuentra en la pág. 15 en **detalles de implementación**.

**CartaDelOponente**

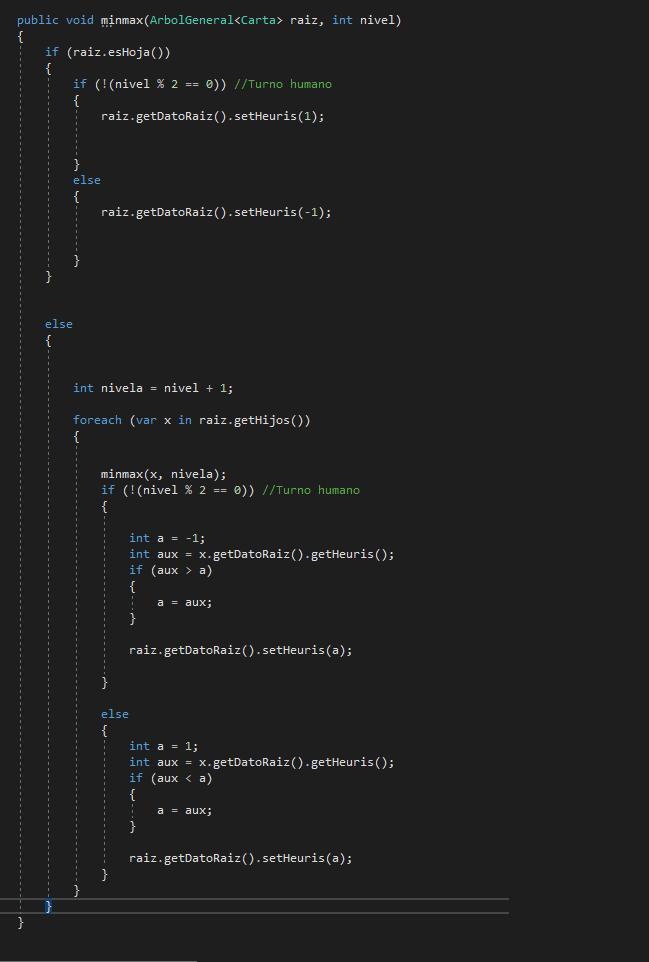
****

**DescartarUnaCarta**

**Detalles de implementación**

Problemas encontrados con la función MinMax:

Parte 1:



Como dijimos, la función **MinMax** consiste en establecer los valores heurísticos (+1 o -1) en cada jugada. Si la jugada tiene un valor “+1” quiere decir que la IA va a ganar en dicha jugada, y si la jugada tiene un valor “-1” quiere decir que el Humano va a ganar en dicha jugada.

Para probar si el algoritmo funcionaba correctamente primero empecé con un juego de solo 4 cartas, por lo tanto, el árbol generado seria de un tamaño pequeño. El algoritmo es recursivo y se centra el llegar a las hojas del árbol, una vez encontrada una hoja va a comprobar si dicha hoja se encuentra en el nivel de la IA o del Humano. Si es el nivel del humano quiere decir que fue su turno y perdió, por lo tanto, tendrá un valor heurístico de +1, pero si es el nivel de la IA quiere decir que fue su turno y perdió, por lo tanto, tendrá un valor heurístico de -1.

Ahora solo queda ir volviendo de la recursión, donde se comprobará si la raíz actual se encuentra en el nivel de la IA o del Humano. En caso de ser el turno del Humano lo que se hará es pedirle el valor heurístico de cada hijo de la raíz actual donde estamos parados y guardarlo en una variable llamada “aux”, además tendremos una variable “a” que es igual a “-1”. Recordemos que la IA siempre juega a ganar, entonces el valor heurístico que nos interesa buscar en los hijos es “+1”.

Entonces, si “aux” es mayor a “a” quiere decir que un hijo tiene el valor “+1”, por lo tanto, cambiaremos el valor de “a” por “+1”. Luego a la raíz actual le seteamos su valor heurístico pasándole por parámetro “a”, de esta forma tendrá el valor “+1” en caso de haber encontrado un hijo con dicho valor, y si no encontró un hijo con el valor “+1” entonces la raíz actual tendría el valor heurístico “-1”.

Esto funciona bien con 4 cartas en el juego, pero si hacemos que el juego tenga 6 cartas entonces los valores heurísticos de cada jugada será incorrecto. Esto se debe a la siguiente línea de código:

if(aux>a)

{

a=aux;

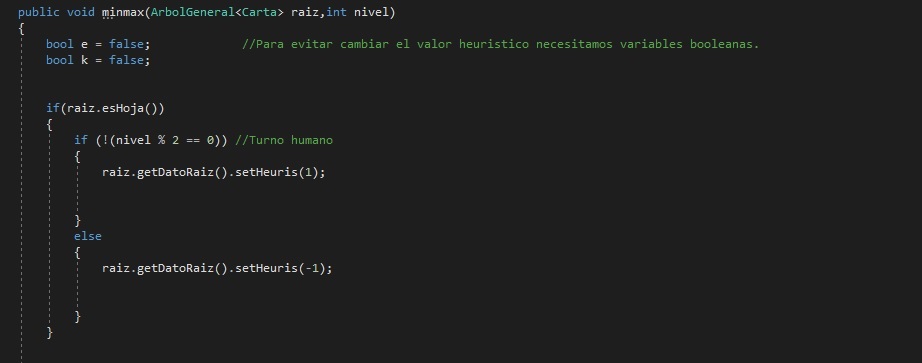
}

raiz.getDatoRaiz().setHeuris(a);

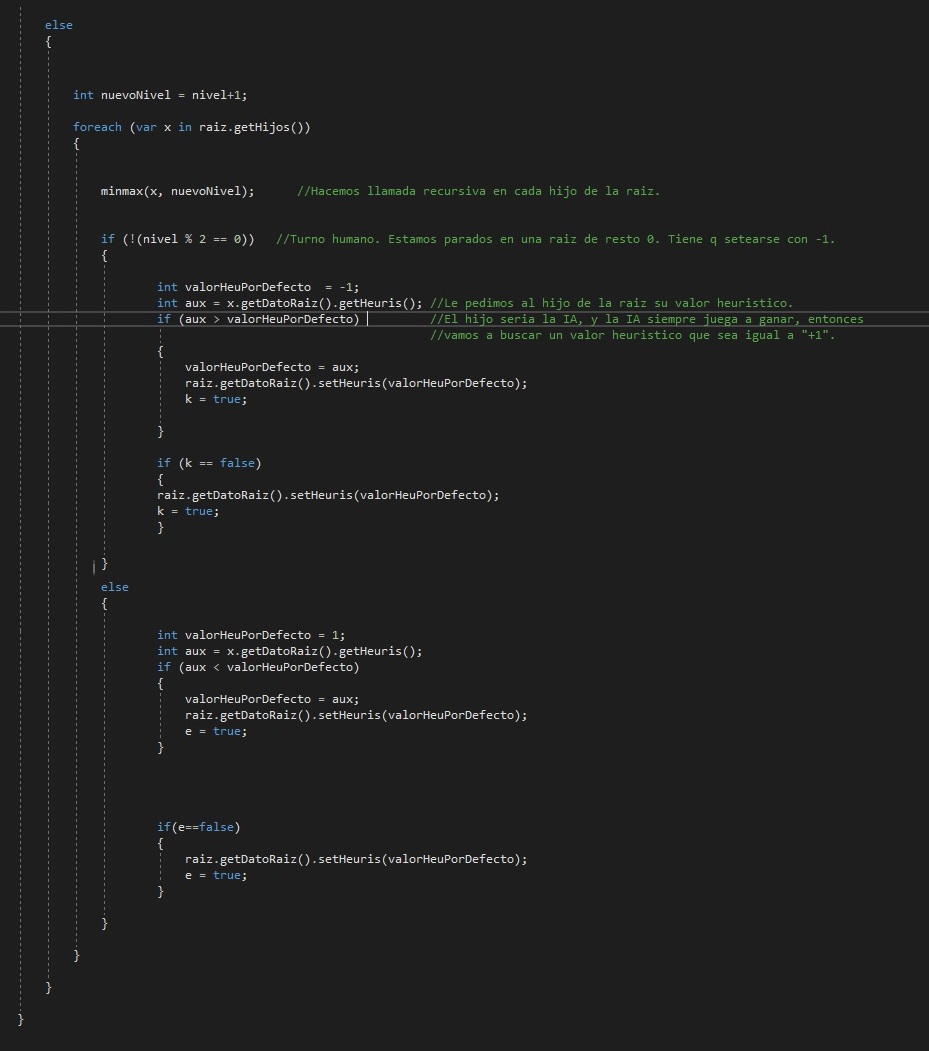
El problema es que si por ejemplo el primer hijo tiene el valor heurístico “+1”, pero el resto de hijos tiene el valor heurístico “-1”, entonces al final la raíz actual tendrá el valor heurístico “-1”, y esto esta mal ya que nosotros queremos el valor “+1” porque la IA siempre juega a ganar.

Para solucionarlo haremos lo siguiente:

Parte 2:



Crearemos dos variables booleanas que estén fuera del “**if**” y el “**else**”. Estas variables nos ayudaran a evitar que se cambien los valores heurísticos de cada carta.



Ahora la variable “k” está en “false”, por lo tanto, supongamos que el primero hijo tiene el valor heurístico “+1”, entonces Aux será igual a “+1”. La variable “valorHeuPorDefecto” es igual a “-1”. Luego tenemos un **if** donde la condición es si “aux” es mayor a “valorHeuPorDefecto”, por lo tanto, entrara al **if**. La variable “valorHeuPorDefecto” será igual a aux, por lo tanto, será igual a “+1”. Luego se seteara el valor heurístico de la raíz actual pasándole por parámetro “valorHeuPorDefecto”, y la variable “k” será igual a “true”, por lo tanto, ahora la raíz actual tiene el valor heurístico “+1” y no se verá modificado por el resto de hijos ya que para que suceda eso “k” tendría q ser igual a “false”.

**Una breve conclusión o reflexión de la experiencia adquirida a partir de la realización del trabajo final.**

La experiencia adquirida en este trabajo fue muy buena, poder usar árboles y recursión me permitió asentar mucho mejor los conceptos y entendimiento de estos. La idea de hacer un videojuego con una IA la cual debe decidir la mejor jugada fue bastante interesante, logrando motivarme a implementarlo, hacer que funcione correctamente, y aprender cómo funciona un árbol con el algoritmo miniMax.