# Conecta-4

Sofía Almeida Bruno

Jesús Sánchez de Lechina Tejada

## Índice

# Conecta 4

## Introducción

Esta memoria recoge todos los datos necesarios para la comprensión del TDA **Conecta4** y el conjunto de operaciones que implementa para permitir la realización de una partida de este clásico juego contra un jugador automático.

## Cambios en el código fuente proporcionado

### Tablero

En tablero los cambios se enfocan principalmente a un constructor que nos permita elegir quién empezará el juego y a la distinción en caso de empate. Así, como los correspondientes datos miembro para permitir estas comprobaciones.

### Datos miembro

Han sido añadidas MAX_PIEZAS y colocadas para tener un control que nos permita determinar los casos de empate.

```
const int MAX_PIEZAS   ///< Máximo de piezas posibles a introducir
int colocadas          ///< Número de piezas introducidas en cada momento
```

### Métodos de la clase

#### Constructor con turno:

Crea un tablero introduciendo el tamaño del mismo. El estado inicial del tablero es todo 0, es decir, todo el tablero está libre. El turno inicial se fijará según el parámetro introducido.

En tablero.hpp:

```
/**
    * @brief Constructor. Crea un tablero introduciendo el tamaño del mismo.
    *      El estado inicial del tablero es todo 0, es decir, todo el tablero
    *      está libre. El turno inicial es el del jugador indicado.
    * @param filas : Número de filas que tendrá el tablero.
    * @param columnas : Número de columnas del tablero.
    * @param turno : Turno del jugador que comenzará insertando
*/

Tablero(const int filas, const int columnas, const int turno);
```

En tablero.cpp:

```
Tablero::Tablero(const int filas, const int columnas, const int turn ) :
  filas(filas), columnas(columnas), MAX_PIEZAS(filas*columnas), colocadas(0), turno(turn) {
    reserve();
}
```

#### Comprobar empate:

Hace uso del dato miembro *MAX_PIEZAS* y del número de fichas colocadas para comprobar si quedan huecos en el tablero. En caso de no quedar huecos y de no haber un ganador se ha producido un empate.

En tablero.hpp:

```
    /**
     * @brief Comprueba si se ha producido un empate
     * @return True si hay empate
     **/
    bool hayEmpate();
```

En tablero.cpp:

```
bool Tablero::hayEmpate() {
  if(colocadas == MAX_PIEZAS && quienGana() == 0)
    return true;
  else
    return false;
}
```

**Incrementar colocadas:**

Debido a la implementación del turno automático, que devuelve el tablero elegido y lo asigna al tablero de la partida en lugar de insertar propiamente en una columna, queda la necesidad de aumentar el número de fichas en el turno automático para seguir controlando los casos de empate.

En tablero.hpp:

```
    /**
     * @brief Aumenta el número de fichas colocadas
     **/
    void incrementaColocadas() {colocadas++;}
```

### Mando

En la clase Mando el único cambio realizado ha sido la implementación de un método que actualice el mando sin insertar ninguna ficha. Lo hemos implementado de esta forma para solucionar un bug que mostraba la ficha errónea en el turno humano hasta que se actualizaba de nuevo el tablero.

En mando.hpp:

```
  /**
   * @brief Actualiza el tablero del juego: la posición del jugador y si se
   *      ha introducido alguna pieza en el tablero.
   *      Esta función es la encargada de controlar la entrada del teclado.
   * @param c : Caracter leído por el teclado.
   * @param t : Tablero. Si el caracter leido es el de colocar ficha y hay hueco
   *      modifica el tablero.
   * @return Devuelve true si se ha colocado una ficha en la actualización del
   *      juego.
   */
  bool actualizarJuego(char c, Tablero & t);
```

En mando.cpp:

```
bool Mando::actualizarAuto(Tablero & t) {
  mando.at(posicion+1) = '^';
  if(t.GetTurno() == 1) jugador.at(posicion+1) = 'x';
  else jugador.at(posicion+1) = 'o';
}
```

**ArbolGeneral**

Esta clase no ha sido modificada respecto al código proporcionado original

## TDA Conecta4

En conecta4, como ya hemos explicado antes, se desarrollan los métodos necesarios para almacenar todos los tableros posibles y la manera de manejar estos para su evaluación.

### Datos miembro

La parte privada de esta clase consta del árbol de posibilidades, la métrica que definimos al crear el TDA y una constante MAX_DEPTH que se encargará de controlar la profundidad máxima del árbol de posibilidades.

Parte privada en conecta4.hpp:

```
 private:
  ArbolGeneral<Tablero> arbol_posibilidades; ///< Árbol que almacena los tableros
  int metrica_elegida; ///< Número de métrica que evaluará los tableros
  const int MAX_DEPTH = 3; ///< Número de niveles a comprobar
```

### Métodos públicos

Los métodos de esta clase se dedican a seleccionar y devolver el tablero correspondiente. A continuación procederemos a detallarlos:

**Constructor tablero y métrica:**

Dado un tablero, crea el TDA con su correspondiente árbol de posibilidades y métrica.

En conecta4.hpp:

```
  /**
   * @brief Constructor
   * @param tab Tablero inicial
   * @param met Métrica a utilizar durante la partida
   */
  Conecta4(const Tablero& tab, int met);
```

En conecta4.cpp:

```
Conecta4::Conecta4(const Tablero& tab, int met) {
  metrica_elegida = met;
  arbol_posibilidades = ArbolGeneral<Tablero>(tab);
  generar_arbol_posibilidades(arbol_posibilidades.raiz(), 0);
}
```

**Obtener árbol de posibilidades:**

Usado principalmente en pruebas, nos permite obtener el árbol para estudiarlo.

En conecta4.hpp:

```
  /**
   * @brief Devuelve el árbol de posibilidades
```

```
   * @return Dato miembro arbol_posibilidades
   */
  ArbolGeneral<Tablero> get_arbol_posibilidades() const {return arbol_posibilidades;}
```

**Generar el árbol de posibilidades:**

Como su propio nombre indica, genera el árbol de posibilidades en base a un nodo indicado como argumento. De manera recursiva, vamos generando un árbol que en cada nivel tiene todas las opciones de juego de uno de los jugadores y en el siguiente las del contrario.

En conecta4.hpp:

```
/**
   * @brief generamos el árbol de posibilidades
   * @param raiz raíz del árbol a generar
   * @param profundidad en la que se encuentra dicho nodo
   * @pre El nodo a partir del cual queremos generar el árbol, debe estar ya en el árbol
   * @return Void
   */
  void generar_arbol_posibilidades(const ArbolGeneral<Tablero>::Nodo& raiz, int profundidad);
```

En conecta4.cpp:

```
void Conecta4::generar_arbol_posibilidades(const ArbolGeneral<Tablero>::Nodo& raiz, int profun
  profundidad++;
  // Primero aumentamos la profundidad, pues acabamos de descender a un hijo
  // (en caso de ser la raíz pasamos del nivel 0 al 1)
  if (profundidad > MAX_DEPTH)
  // Si ya estamos a la máxima profundidad no seguimos generando hijos
    return;

  int columnas = arbol_posibilidades.etiqueta(raiz).GetColumnas();
  ArbolGeneral<Tablero> aux;
  Tablero hijo(arbol_posibilidades.etiqueta(raiz));

  int i;
  bool insertado = false;
  /* Primero se inserta en el hijo izquierda y luego insertamos en los hermanos derecha.
     Por lo tanto, primero tenemos que buscar el primer hueco.
     Si lo hay, usamos insertar_hijomasizquierda(), pero para los siguientes
     hijosraiz tenemos que insertar como hermanos_derecha del hijo que ya
     tenemos, por eso usamos esta condición
     para distinguir el tipo de inserción que tenemos que hacer*/

  // Para el hijo más a la izquierda
  for (i = 0; i < columnas && !insertado; ++i) {
    if (arbol_posibilidades.etiqueta(raiz).hayHueco(i) != -1) {
      hijo.cambiarTurno();
      insertado = hijo.colocarFicha(i);
      aux.AsignaRaiz(hijo);
      arbol_posibilidades.insertar_hijomasizquierda(raiz, aux);
```

```
      generar_arbol_posibilidades(arbol_posibilidades.hijomasizquierda(raiz), profundidad);
    }
  }
```

**Actualizar árbol:**

Volvemos a generar el árbol de posibilidades a partir del tablero que está en ese momento en juego.

En conecta4.hpp:

```
  /**
   * @brief Actualiza el árbol de posibilidades, dejando como raíz el tablero pasado
            y completando el árbol hasta la profundidad MAX_DEPTH
   * @param tablero actual, a partir del cual se regenera el árbol
   */
  void actualizar(const Tablero& tablero);
```

En conecta4.cpp:

```
void Conecta4::actualizar(const Tablero& tablero) {
  arbol_posibilidades.AsignaRaiz(tablero);
  generar_arbol_posibilidades(arbol_posibilidades.raiz(), 0);
}
```

**Mejor tablero:**

Busca y devuelve el mejor tablero para el jugador automático según la métrica que usemos.

En conecta4.hpp:

```
  /**
   * @brief Devuelve el mejor tablero
   * @return tablero a utilizar por el jugador automático
   */
  Tablero mejor_tablero(Tablero tablero);
```

En conecta4.cpp:

```
Tablero Conecta4::mejor_tablero(Tablero tablero) {
  if(metrica_elegida == 1) {
    return metrica1(tablero);
  }
  else {
    if(metrica_elegida == 2)
      return metrica2(tablero);
    pair<ArbolGeneral<Tablero>::Nodo, int> p;
    p = recorrer_arbol(arbol_posibilidades.raiz());
    return arbol_posibilidades.etiqueta(p.first);
  }
}
```

**Recorrer tablero:**

Devuelve un pair con el mejor tablero y su puntuación, esta función se realiza de manera recursiva.

En conecta4.hpp:

```
/**
   * @brief Recorre el árbol devolviendo el mejor tablero
   * @param raiz Nodo que será la raíz del subárbol a recorrer
   * @return un pair formado por el mejor tablero (en forma de nodo) y su puntuación
   */
  pair<ArbolGeneral<Tablero>::Nodo, int>
        recorrer_arbol(const ArbolGeneral<Tablero>::Nodo& raiz);
```

En conecta4.cpp:

```
pair<ArbolGeneral<Tablero>::Nodo, int>
    Conecta4::recorrer_arbol(const ArbolGeneral<Tablero>::Nodo& raiz) {

  if(arbol_posibilidades.altura(raiz) == 0) {
    pair<ArbolGeneral<Tablero>::Nodo, int> p(raiz,
        metrica(arbol_posibilidades.etiqueta(raiz)));

    return p;
  }
  else {
    // Resto de casos
    // Para cada hijo llamo a recorrer_arbol
    pair<ArbolGeneral<Tablero>::Nodo, int> maximo =
        recorrer_arbol(arbol_posibilidades.hijomasizquierda(raiz));
    maximo.second += metrica(arbol_posibilidades.etiqueta(raiz))
        * arbol_posibilidades.altura(raiz);

    if(arbol_posibilidades.altura(raiz) != 1) {
      maximo.first = arbol_posibilidades.hijomasizquierda(raiz);
    }
    pair<ArbolGeneral<Tablero>::Nodo, int> intermedio(maximo);

    ArbolGeneral<Tablero>::Nodo  nodo_aux;

    while((arbol_posibilidades.hermanoderecha(intermedio.first)) != NULL) {
      nodo_aux = arbol_posibilidades.hermanoderecha(intermedio.first);

      intermedio = recorrer_arbol(nodo_aux);
      intermedio.first =  nodo_aux;
      intermedio.second += metrica(arbol_posibilidades.etiqueta(nodo_aux))
          * arbol_posibilidades.altura(nodo_aux);

      if(maximo.second < intermedio.second)
          maximo = intermedio;
    }

    return maximo;
```

```
  }
}
```

**Selección de métrica:**

Controla el flujo de ejecución del programa, pues según la métrica elegida se busca un tablero determinado u otro.

En conecta4.hpp:

```
  /**
   * @brief Si la métrica es la 3 o la 4, esta función se encarga de devolver la
   *        puntuación del tablero según la métrica escogida
   * @param tablero Tablero a evaluar
   * @return Puntuación del tablero
   */
  int metrica(Tablero &tablero);
```

En conecta4.cpp:

```
int Conecta4::metrica(Tablero &tablero) {

  int puntuacion;

  switch (metrica_elegida) {
  case 3:
    puntuacion = metrica3(tablero);
    break;
  case 4:
    puntuacion = metrica4(tablero);
    break;
  default:
    cout << "Error en selección de métrica para inserción." << endl;
    exit(-1);
  }

  return puntuacion;
}
```

**Mostrar árbol en preorden:**

Muestra el árbol de posibilidades en preorden. Usado meramente con fines de depuración.

En conecta4.hpp:

```
/**
   * @brief Muestra el recorrido en preorden del árbol de posibilidades
   * @return Void
   */
  void mostrar_arbol_preorden() {arbol_posibilidades.recorrer_preorden();}
};
```

### Métricas usadas

Esta sección, si bien sigue perteneciendo al código del TDA conecta4, es importante distinguirla del resto del código.

**Métrica 1:**

*"La mejor métrica"*, su prioridad es ganar si consigue alinear fichas, a su vez cortará toda victoria directa del adversario, en su defecto inserta aleatoriamente. Devuelve directamente el tablero a escoger.

En conecta4.hpp:

```
/**
   * @brief Mejor métrica, intenta alinear cuatro, en caso contrario,
            cortar las victorias rivales; si esto tampoco es posible,
            se inserta aleatoriamente
   * @param tablero Tablero a evaluar
   * @return Tablero correspondiente
   **/
  Tablero metrica1(Tablero &tablero);
```

En conecta4.cpp:

```
Tablero Conecta4::metrica1(Tablero &tablero) {
  int cols = tablero.GetColumnas();

  //Comprobamos si podemos ganar
  for(int i = 0; i < cols; ++i) {
    Tablero aux(tablero);
    if(aux.colocarFicha(i)) {
      if(aux.quienGana() == 2) {
    tablero.colocarFicha(i);
    return tablero;
      }
    }
  }

  //Cortamos jugadas ganadoras
  for(int i = 0; i < cols; ++i) {
    Tablero aux(tablero);
    aux.cambiarTurno();
    if(aux.colocarFicha(i)) {
      if(aux.quienGana() == 1) {
    tablero.colocarFicha(i);
    return tablero;
      }
    }
  }
```

```
  //Insertamos aleatoriamente
  bool insertada = false;
  do {
    int pos = rand() % cols;
    if(tablero.colocarFicha(pos))
      return tablero;
  } while(!insertada);
}
```

**Métrica 2:**

Con una mentalidad similar a la anterior se limitará a cortar cualquier victoria rival que se le presente, si no puede encontrarla recurre a la inserción aleatoria. Devuelve un tablero.

En conecta4.hpp:

```
/**
   * @brief Segunda métrica, corta la victoria rival.
           Si no fuera posible, llama a la métrica 3
   * @param tablero Tablero a evaluar
   * @return Tablero correspondiente
   **/
  Tablero metrica2(Tablero &tablero);
```

En conecta4.cpp:

```
Tablero Conecta4::metrica2(Tablero &tablero) {
  int cols = tablero.GetColumnas();

  //Cortamos victorias del contrario
  for(int i = 0; i < cols; ++i) {
    Tablero aux(tablero);
    aux.cambiarTurno();
    if(aux.colocarFicha(i)) {
      if(aux.quienGana() == 1) {
    tablero.colocarFicha(i);
    return tablero;
      }
    }
  }

  //Insertamos aleatoriamente
  bool insertada = false;
  do {
    int pos = rand() % cols;
    if(tablero.colocarFicha(pos))
      return tablero;
  } while(!insertada);
}
```

**Métrica 3:**

Combinada con recorrer_arbol, busca la victoria directa en tres niveles de profundidad y procurará seguir aquellas ramas que lleguen a una victoria. También intenta evitar las ramas donde el contrincante gana, pero está orientado a la victoria en lugar de a la defensa. Devuelve la puntuación del tablero, que será mayor si produce una victoria en el turno siguiente.

En conecta4.hpp:

```
/**
   * @brief Métrica que sólo verifica si se produce una victoria.
            En caso de no victoria introducirá una puntuación aleatoria
   * @param tablero Tablero
   * @return pair tablero y puntuación máxima
   **/
  int metrica3(Tablero &tablero);
```

En conecta4.hpp:

```
int Conecta4::metrica3(Tablero &tablero) {
  if (tablero.quienGana() == 0)
    return metrica4(tablero);

  if (tablero.quienGana() == 2)
    return 1000;

  else
    return -1500;
}
```

**Métrica 4:**

Asigna puntuaciones aleatorias entre los 3 siguientes niveles y decide. Esta métrica, como la anterior, devuelve una puntuación en lugar de un tablero, para decidir cual escoger.

En conecta4.hpp:

```
/**
   * @brief Otorga puntuaciones aleatorias entre 0 y 100
   * @param tablero en el que insertará
   * @return pair tablero y puntuación máxima
   */
  int metrica4(const Tablero &tablero);
```

En conecta4.cpp:

```
int Conecta4::metrica4(const Tablero& tablero) {
  int puntuacion = rand() % 100;
  return puntuacion;
}
```

### Main

En main principalmente hacemos una bifurcación en el programa de acuerdo a la selección del tipo de partida y los parámetros introducidos: Partidas 1vs.1 o 1vs.IA

En **jugar_partida_humanos** añadimos la distinción del caso de empate, que no era considerada.

```
int jugar_partida_humanos(int filas = 5, int columnas = 7) {

  Tablero tablero(filas, columnas);      //Tablero filxcols
  Mando mando(tablero);       //Mando para controlar E/S de tablero
  char c = 1;
  int quienGana = tablero.quienGana();
  //mientras no haya ganador y no se pulse tecla de terminación
  while(c != Mando::KB_ESCAPE && quienGana == 0) {
    system("clear");

mando.actualizarJuego(c, tablero);  // actualiza tablero según comando c
    imprimeTablero(tablero, mando);     // muestra tablero y mando en pantalla
    quienGana = tablero.quienGana();    // hay ganador?

if(tablero.hayEmpate()) {
      return -2;
    }

if(quienGana==0) c = getch();       // Capturamos la tecla pulsada.
  }

  return tablero.quienGana();
}
```

En **jugar_partida**, humano contra inteligencia artificial se distingue entre turno humano y turno automático.

En el turno humano nos aseguramos de que sólo se inserte ficha una vez elegido una columna válida y que se cambie el turno (de esto se encarga la función actualizarJuego(c,tablero) de la clase *Mando*).

Por el otro lado el jugador automático simula el pensamiento de la máquina "*durmiendo*" un segundo, para luego elegir el mejor tablero y asignarlo. Dado que no insertamos en una columna sino que asignamos al tablero en juego debemos de manualmente incrementar el número de fichas colocadas y cambiar el turno. Por último llamamos a actualizarAuto() para que cuando vuelva a ser el turno humano se imprima correctamente el tablero al comienzo del turno.

```
int jugar_partida(int filas = 4, int columnas = 4, int metrica = 1, int turno = 1) {
  //(filas, columnas, metrica, turno)

  Tablero tablero(filas, columnas, turno);      //Tablero filas x columnas
  int quienGana = tablero.quienGana();
  char c = 1;
```

```
  Conecta4 j_auto(tablero, metrica);
  Mando mando(tablero);
  bool insertado;

  //Mientras no haya ganador y no se pulse la tecla de terminación
  while(c != Mando::KB_ESCAPE && quienGana == 0) {
    system("clear");

    if(tablero.GetTurno() == t_humano) {
      cout << "Jugador humano" << endl;
      imprimeTablero(tablero, mando); //Muestra tablero y mando en pantalla
      c = getch(); //Capturamos la tecla pulsada
      insertado = mando.actualizarJuego(c, tablero);
      if (insertado) {
    //No queremos actualizar el árbol de posibilidades si no se insertó ficha
    tablero.cambiarTurno();
    j_auto.actualizar(tablero);
    tablero.cambiarTurno();
      }
    }
    else {
      cout << "Jugador automático" << endl;
      imprimeTablero(tablero,mando);
      sleep(1);
      tablero = j_auto.mejor_tablero(tablero);
      tablero.incrementaColocadas();
      tablero.cambiarTurno();
      mando.actualizarAuto(tablero);
    }

    system("clear");
    quienGana = tablero.quienGana();
    if(tablero.hayEmpate()) {
      system("clear");
      imprimeTablero(tablero, mando);
      return -2;
    }

  }
  system("clear");
  imprimeTablero(tablero, mando);
  return tablero.quienGana();
}
```