Universidad Rey Juan Carlos

Escuela Superior de Ciencias Experimentales y Tecnología

jReversi

Inteligencia Artificial

Proyecto 3: búsqueda antagonista

Angel Jara Gómez
Jaime Pérez Crespo
Tomás Aguado Gómez

Introducción

Los juegos son uno de los campos de acción más antiguos de la Inteligencia Artificial; por lo general es muy sencillo representar el estado del juego, y las acciones que cada jugador (humano o máquina) puede realizar están lo suficientemente restringidas para que el problema sea computacionalmente abordable.

Sin embargo, incluso para juegos muy sencillos, la cantidad de opciones posibles hace que sea excesivamente difícil resolverlos: los árboles de búsqueda completos se hacen demasiado grandes y resolver el problema manteniendo la interactividad con el jugador humano se hace imposible.

Es por esto que aparece un nuevo tipo de incertidumbre, no debida a la falta de información, sino aparejada a la imposibilidad de averiguar todas las consecuencias posibles de una jugada hasta la finalización de la partida.

Consideremos el caso general de un partida con dos jugadores, (*MAX* y *MIN*). *MAX* comienza a jugar y ambos van alternando los turnos de juego hasta la finalización de la partida. Si se tratara de un problema de búsqueda normal, *MAX* sólo tendría que buscar en su árbol de búsqueda una secuencia de jugadas que le llevara a un estado ganador.

Sin embargo el otro jugador (*MIN*) tomará decisiones que redunden en su beneficio y, por tanto, en perjuicio de *MAX*.

El algoritmo *minimax* sirve para determinar la decisión óptima que debería tomar *MAX* (el primer jugador, el que tiene el turno) suponiendo que el jugador *MIN* actuará coherentemente para ganar eligiendo su mejor jugada.

El algoritmo *minimax* básico parte de la suposición de que el programa dispone de todo el tiempo necesario para efectuar la búsqueda anteriormente mencionada hasta llegar a los nodos “hoja”, o estados terminales del juego, donde la partida ha terminado y uno de los dos participantes ha ganado o se ha producido un empate si lo permite la lógica del juego. Este hecho se representa usando una función de utilidad que se puede representar con los valores *+1*, *-1* o *0* en caso de empate.

Como hemos dicho ya, este planteamiento haría que el programa, en este caso un juego, exigiera costes tanto de tiempo de computación como de memoria que lo hacen inabordable a efectos prácticos.

Es por esto que el algoritmo usado para el juego *jReversi* es *minimax* con poda *Alfa-Beta*. Esta versión del algoritmo introduce varios cambios respecto al modelo simple. En primer lugar no será necesario llegar a los estados terminales del juego, sino que se definirá una profundidad máxima de búsqueda (directamente relacionada con la dificultad de juego que selecciona el usuario) y

consideraremos como terminales los nodos que estén a esa profundidad o los que (como en la versión básica del algoritmo) supongan el fin de la partida. También cambiaremos la función de utilidad por una función de evaluación heurística más compleja que evalúe "cómo de bueno" es cada uno de estos nodos terminales. Para un nodo terminal real (esto es, uno en el que se alcance el final del juego) se comportará como la función de utilidad y para un nodo terminal definido por la profundidad devolverá un valor intermedio en función del estado del juego en ese nodo.

Ahora bien, teniendo juegos como el ajedrez en los que tenemos un tiempo limitado para realizar la jugada las cosas empiezan a complicarse. Se hace necesario evitar la exploración (expansión) de ramas del arbol cuyo análisis no nos vaya a reportar resultados nuevos. A este mecanismo se le llama poda.

Gracias a la poda obtenemos la misma jugada que obtendríamos usando el algoritmo *minimax* simple pero eliminando las ramas que no van a influir en la decisión final.

Siendo *alfa* el valor de la mejor opción encontrada hasta entonces a través de la ruta de *MAX* y *beta* el valor más favorable (más bajo) a través de la ruta de *MIN*, según recorremos el árbol recalculamos los valores de *alfa* y *beta* y si los de una rama concreta son peores que los valores que tenemos almacenados, automáticamente podaremos esa rama.

Para mejorar este algoritmo, como parece lógico, hay que maximizar el número de ramas que se podan. Luego tendremos que examinar en primer lugar las ramas que parecen mejores.

La debilidad del algoritmo *minimax* con poda *alfa-beta* no está en el algoritmo en sí sino en la función de evaluación. Toda la eficacia de *minimax* se basa en el valor que ofrece la función de evaluación de un nodo considerado como terminal. En el caso del juego del *Otelo*, podríamos decir, por ejemplo, que una jugada que le deja abierto al oponente el camino hacia una esquina es mucho peor que otra que nos hace ganar una casilla lateral o incluso una esquina.

Sin embargo esta evaluación del estado del juego no es más que una estimación y por muy bueno que sea el algoritmo, si esta función no está bien diseñada, será facil vencer a la máquina una vez que se aprendan las debilidades causadas por un mal diseño de esta función heurística de evaluación.

Heurística

En nuestro caso se ha diseñado esta función basándonos en una matriz donde se asigna a cada casilla un peso específico dentro del juego. Como se ha comentado anteriormente la estrategia que seguirá la máquina será la de apoderarse de las casillas laterales (y en particular, las esquinas) en cuanto le sea posible, ya que son estratégicas para vencer.

De este modo cuando se evalúa un tablero la primera tarea que ha de cumplir la función de evaluación es comprobar si la partida ha terminado y en tal caso, averiguar quién ha ganado. Si quién ganó fue la máquina la función devuelve infinito (es la situación más deseable para la máquina y la menos adecuada para su contrincante). Si por el contrario la máquina pierde la evaluación devuelve infinito negativo, haciendo esa jugada indeseable y aceptable únicamente como última opción para la inteligecia artificial del juego. En caso de empate la función de evaluación devuelve cero.

Otros nodos hoja son los que se alcanzan no como estados terminales del juego, sino como nodos que igualan la profundidad máxima a la se genera el árbol del juego. En este caso ninguno de los dos jugadores ha ganado, y hay que analizar el estado del tablero, asignándole una puntuación en función de lo ventajosa que sea la situación para la máquina.

Esta asignación se basa en una matriz que refleja las tácticas que hemos mencionado anteriomente:

```
{120, -40, 20,  5,  5, 20, -40, 120}
{-40, -60, -5, -5, -5, -5, -60, -40}
{ 20,  -5, 15,  3,  3, 15,  -5,  20}
{  5,  -5,  3,  3,  3,  3,  -5,   5}
{  5,  -5,  3,  3,  3,  3,  -5,   5}
{ 20,  -5, 15,  3,  3, 15,  -5,  20}
{-40, -60, -5, -5, -5, -5, -60, -40}
{120, -40, 20,  5,  5, 20, -40, 120}
```

La función de evaluación recorre el tablero actual aplicando esta matriz como máscara y acumulando la puntuación que indica esta matriz a un contador si la casilla está ocupada por una pieza de la máquina.

Si por el contrario la casilla tiene una pieza del oponente se resta el valor de la casilla al acumulador.

De este modo, un estado del juego es favorable a la máquina por varias razones:

- Porque tenga muchas fichas de su color en el tablero.
- Porque tenga fichas estratégicas en el tablero (laterales y esquinas).

Las casillas centrales tienen una puntuación neutral, de este modo sólo son importantes cuando en una jugada ocupamos varias de las casillas de esta clase. Sin embargo según la situación la máquina cambiará varias fichas centrales por ocupar una casilla lateral o más aun una esquina.

Las casillas vecinas a las esquinas tienen puntuaciones negativas. La razón es que poner una casilla en uno de esos lugares deja abierta al oponente la opción de quedarse con una esquina. Sin embargo esta puntuación tan negativa no tiene sentido si la casilla de la esquina ya está ocupada. Es por eso que cuando se valoran estas zonas se analiza si la esquina está ocupada; si no lo está se procede de la manera usual. Por el contrario si la casilla ya está ocupada se aplica una función de corrección para que estas posiciones tengan un valor similar a cualquiera de las laterales.

Detalles de implementación de la Inteligencia Artificial

A la hora de abordar la implementación del algoritmo *minimax* se hizo desde dos enfoques, originando dos implementaciones diferentes que se adjuntan con el enunciado:

- Mantenemos un árbol con las jugadas posibles. En cada media jugada aumentamos sólo un ply su profundidad. Maximiza la velocidad gracias a la reutilización de cálculos ya hechos, pero consume mucha memoria.
- El árbol con las jugadas se genera cada vez hasta la profundidad definida de manera recursiva. Sólo se mantiene en memoria el camino que se está analizando en ese momento. Es más lento pero el consumo de memoria es mínimo.

En el enfoque recursivo se ha creado una función única *minimax* que se llama a sí misma en lugar de las típicas funciones *MAX* y *MIN* mutuamente recursivas.

La razón ha sido implementar de manera clara una situación que puede darse en el juego y que el algoritmo teórico *minimax* no refleja: la realidad es que a medida que avanza el juego, en las últimas jugadas una mala decisión frecuentemente supone la derrota. En los últimos compases del juego es frecuente que el algoritmo *minimax* encuentre que alguno de los dos jugadores no puede mover. Esta situación no se trata en el algoritmo *minimax* clásico, ya que la función *MAX* sólo puede llamar recursivamente a *MIN* .

Es por esto que las situaciones anteriormente mencionadas no se tratan de manera correcta, pudiendo desecharse ramas que le darían a la *IA* la victoria segura o, por el contrario elegir otras que supondrán una derrota cierta.

En nuestro caso cada nodo puede mutar en *MAX* o *MIN*:

En primer lugar se comprueba si hemos de realizar la suspensión y evaluar directamente el estado del tablero (sea o no terminal) o si la profundidad de exploración ha superado la dificultad que se fijó al inicio de la partida.

De no ser así, se comprueba si se puede expandir para el turno que tiene que mover “por derecho”. Si por la situación del tablero no pudiera mover ese color, el algoritmo intenta expandir hacia el otro color. Si tampoco fuera posible mover el juego habría acabado y habría que evaluar quién es el ganador.

En función del turno para el que se haya hecho la expansión actuamos como si estuviéramos en un nodo *MAX* (turno de la máquina) o en un nodo *MIN* (turno del oponente).

De este modo las situaciones en las que uno de los jugadores puede (y de hecho debe) mover dos veces seguidas quedan recogidas en el algoritmo y evaluadas de manera correcta.

En primera instancia los tiempos de respuesta para el algoritmo (en la versión de consumo reducido de memoria) eran prohibitivos. Sin embargo al implementar las podas *alfa-beta*, el tiempo de búsqueda se redujo enormemente aproximando la interactividad de las dos versiones del algoritmo que se realizaron.

Funcionamiento de la aplicación

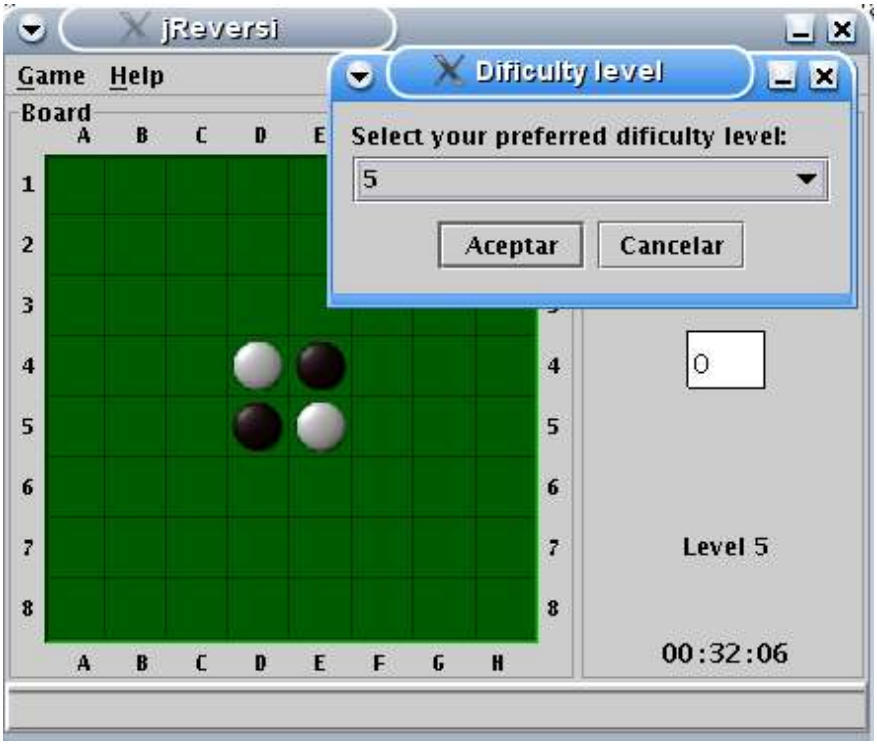
El funcionamiento de la interfaz y el interno del programa se han separado en dos hebras. En primer lugar llamamos al constructor de la clase que contiene todo el comportamiento de la Inteligencia Artificial del programa que estará siempre en este flujo de ejecución.

Una vez llamado a este constructor lanzamos una hebra diferente para que gestione todas las operaciones entre el usuario y la GUI. Esto tiene como objeto evitar bloqueos de la interfaz cuando el programa esta realizando operaciones de alta carga computacional. Por ejemplo si ejecutáramos todo en la misma hebra, un jugador enfrentándose a la máquina en nivel 12 (donde las jugadas tardan mucho más en evaluarse) no podría realizar ninguna operación sobre la interfaz durante el espacio de tiempo en el que la CPU esta analizando la jugada, incluso para el usuario la interfaz daría la impresión de haberse “colgado”.

Lanzando la IA y la GUI en hebras diferentes, el jugador puede abortar y salir del juego en cualquier momento o incluso crear uno nuevo.

Sin embargo al cargar tanto el procesador, a partir de ciertos niveles se aprecia un leve retardo en la operación de la GUI, esto es debido a la voracidad de tiempo de CPU del algoritmo de la IA, que interfiere ligeramente en las operaciones de la GUI.

Una vez lanzadas las dos hebras, la interfaz queda a la espera de una acción del usuario, por ejemplo la de crear un nuevo juego:



Una vez seleccionado el modo de juego y el nivel de dificultad (si procede), entramos en un bucle en el que nos bloqueamos esperando un evento de ratón (movimiento del usuario) para ambos contrincantes (modo dos jugadores), o alternando con llamadas a miniMax si estamos en modo un jugador. Periódicamente se consulta una variable para salir del juego y crear uno nuevo o salir de la aplicación si el usuario lo requiriera.

Detalles de implementación de la Interfaz

A la hora de desarrollar el tablero se optó por un *jTable* sobre el que se desarrolla la partida estando las fichas de ambos jugadores representadas por imágenes. Cuando un jugador "come" una ficha del contrario se ha desarrollado una animación que representa el hecho como el giro de la ficha de un color a otro.

Todo el comportamiento de la interfaz se encapsula en la clase ReversiGUI, que ofrece una API a la clase principal para realizar todas las operaciones sobre el tablero. Cada una de las casillas del panel responde ante el evento de click del usuario siempre que sea su turno. Si es así se comprueba si la jugada que está intentando realizar es válida, en cuyo caso se coloca la ficha del color correspondiente y se voltean las demás.

Situado a la izquierda del panel de juego tenemos las casillas con las puntuaciones, que se han caracterizado como el número de casillas de cada color, así como un reloj para controlar la duración de cada jugada.

El menú consta de los siguiente elementos:

- Menú "GAME":
  - New Game
    Crea un nuevo juego, (interrumpiendo incluso el actual) pidiendo al usuario por pantalla el modo de juego deseado (1 o 2 jugadores) y la dificultad.
  - Player Name
    Define el nombre con el que el usuario jugará la partida. En caso de haber seleccionado una partida de dos jugadores, podrá especificar el nombre de ambos.
  - Quit
    Sale del juego.
- Menú "HELP"
  - About
    Información sobre jReversi.

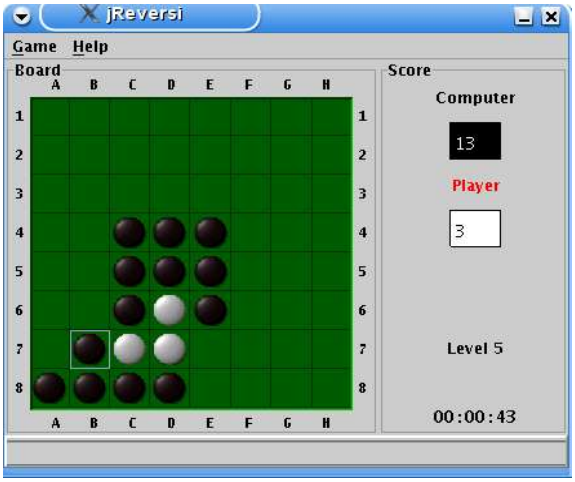
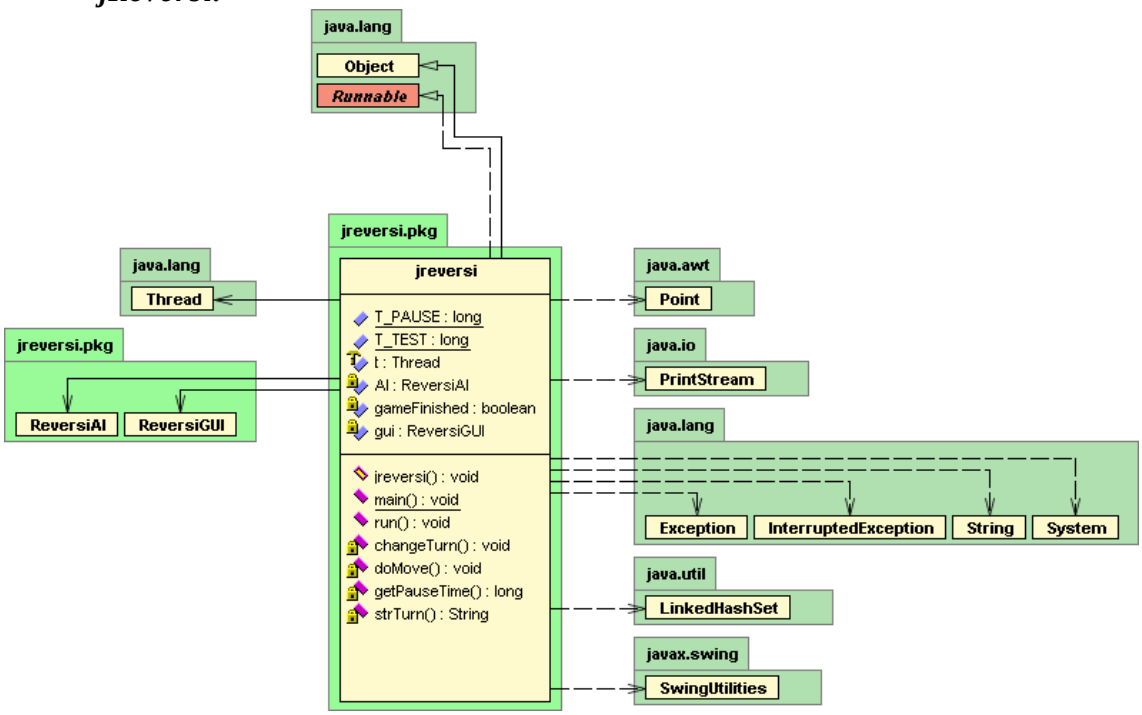
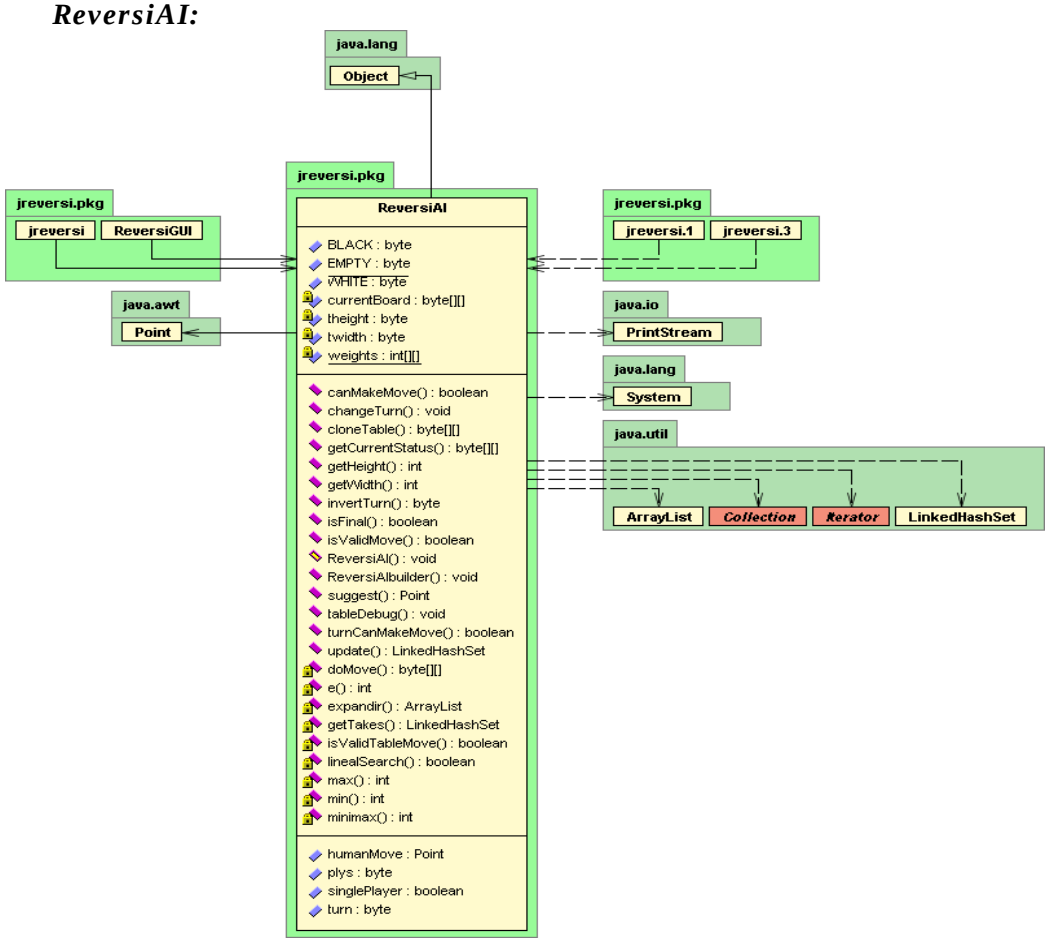


Diagrama de Clases:

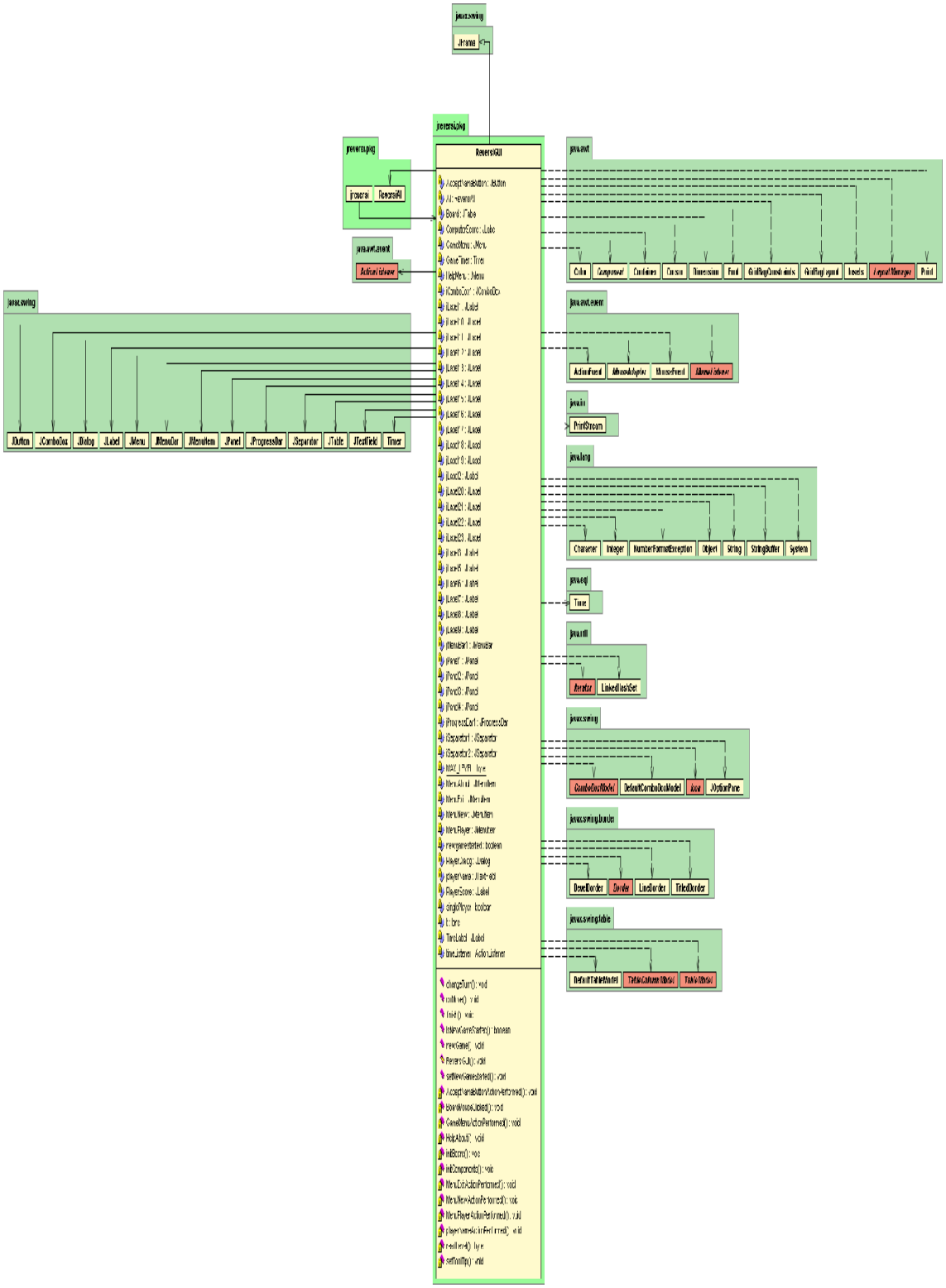
jReversi:



ReversiAI:



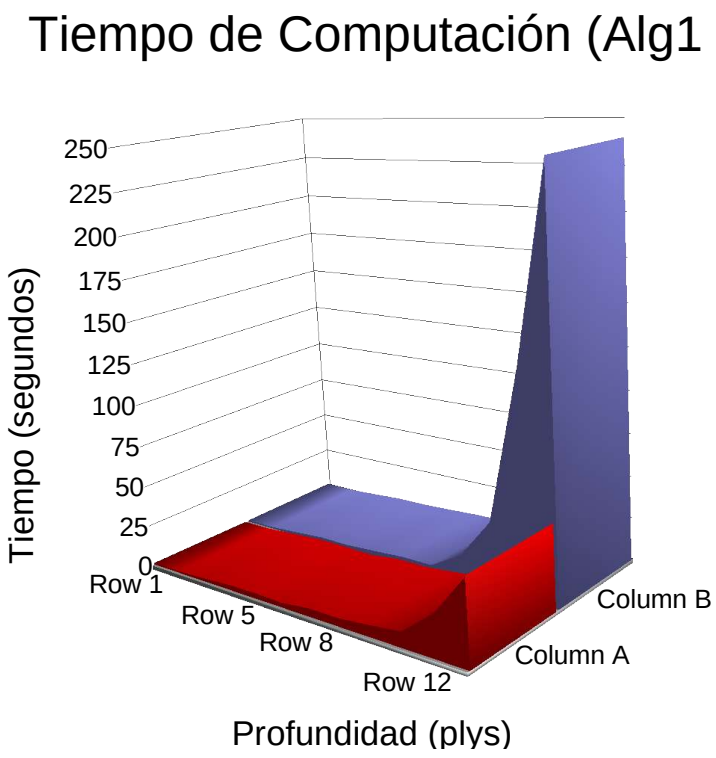
ReversiGUI:



Análisis de ejecución

Como hemos mencionado se han realizado dos implementaciones diferentes: una que mantiene el árbol en memoria (algoritmo 2) para acelerar las búsquedas (consumiendo más memoria), y otra que realiza y explora el árbol cada vez, manteniendo un coste de computación más alto (algoritmo 1).

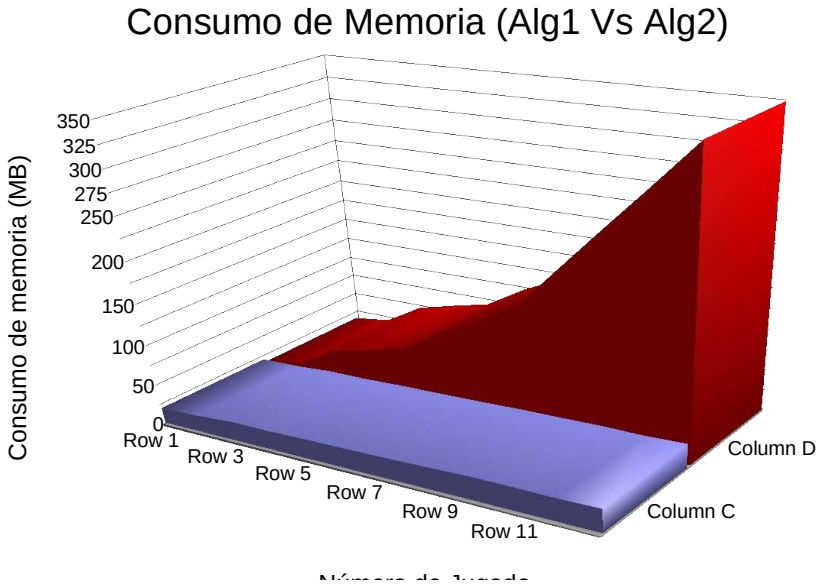
Analizando los tiempos de computación para cada algoritmo:



Como podemos observar el tiempo de computación permanece relativamente bajo hasta el ply 9, a partir de ese momento el árbol crece exponencialmente y así lo hace el tiempo de computación que se usa para generarlo y recorrerlo.

El segundo algoritmo tarda mucho menos en resolver la jugada. Sin embargo en la primera jugada tiene que contruir el árbol completo, por lo que el coste de realizar el primer árbol será similar al de una jugada del algoritmo 1. En general, la velocidad de cálculo será mucho mayor si la situación del juego no lleva a recalcular el árbol con mucha frecuencia.

Sin embargo el consumo de memoria para el algoritmo 2 se hace crítico. Podemos analizar cómo crece dicho consumo a razón de las jugadas que se realizan para dificultad 9, por ejemplo:



El consumo de memoria crece con el número de jugadas posibles. Hay un punto durante el juego en el que el número de jugadas empieza a decrecer (el tablero se va llenando) y el consumo de memoria decrece.

Herramientas utilizadas

jRevesi ha sido desarrollado usando el lenguaje java y el IDE para construcción de interfaces *Netbeans*. Asimismo para el soporte de trabajo distribuido se ha creado un repositorio de *subversion*, una herramienta para el control de versiones derivada de *cvs*. Todas las herramientas usadas para el desarrollo son abiertas, gratuitas y/o de libre distribución.

El uso de Java ha posibilitado obtener un código extremadamente simple y muy fácil de entender, simplificando y potenciando el trabajo colaborativo desarrollado por el grupo.

Netbeans es un IDE de desarrollo de interfaces en Java. Se trata de una herramienta muy avanzada que ofrece soporte para características como la integración del projecto *jReversi* con un repositorio de *subversion*. Al ser multiplataforma, ha permitido desarrollar el proyecto simultáneamente en diferentes plataformas, que incluyen Windows XP, GNU/Linux y Mac OS X.

Subversion es un sistema de control de versiones libre, que ha posibilitado el trabajo asíncrono tanto en tiempo como en espacio de los integrantes del grupo. Cada miembro del equipo antes de ponerse a trabajar se baja la última versión de este repositorio, y pasa a resolver los posibles conflictos (caso de existir). Una vez resueltas puede escribir su propio código. Tras comprobar que su trabajo compila, y que no ha hecho inestable el código con las funcionalidades existentes, pasa a hacer un "*commit*" de sus cambios en el repositorio.

Gimp-2.0 se utilizó para los gráficos y animaciones del tablero de juego.

Instrucciones de instalación y uso

Requisitos para Windows XP:

- *J2SE Development Kit* (*JDK*) *5.0 update 4* (necesario para compilar/ejecutar la aplicación).
- *J2SE Runtime Environment* (*JRE*) *5.0 update 4* (necesario para ejecutar la aplicación).

Para ejecutar la aplicación sólo es necesario JRE.

Instrucciones de ejecución:

- Windows XP: ejecutar `jreversi.bat`.
- GNU/Linux / Mac OS X / otros UNIX: ejecutar `jreversi.sh`.
- Cualquier sistema: desde una línea de comandos, ejecutar `java -jar jreversi.jar`.

Puede ser necesario actualizar la variable de entorno `PATH`, incluyendo la ruta al ejecutable java:

```
set PATH=%PATH%;/ruta/al/ejecutable/java
```

En el caso de la versión 5.0-4 de *JRE*, esta ruta es "`C:\Archivos de programa\Java\jdk1.5.0_04\bin`". Si la versión de *JRE* no es la 5.0-4, puede haber problemas al ejecutar la aplicación. En tal caso, puede solucionarse compilando (suponiendo que la versión de *JRE* del usuario no sea demasiado antigua).

Instrucciones de compilación:

- Windows XP: ejecutar `compilar.bat`.
- GNU/Linux / Mac OS X / otros UNIX: ejecutar `compilar.sh`.
- Cualquier sistema: desde una línea de comandos, ejecutar:

```
javac *.java
jar cmf Manifest jreversi.jar *.class
```

Código fuente (versión 1, sin reutilización de cálculo)

Código Fuente de la clase principal Jreversi:

```
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class jreversi implements Runnable {

    public static final long T_TEST = 100;
    public static final long T_PAUSE = 1000;

    private boolean gameFinished = false;
    private ReversiAI AI;

    private ReversiGUI gui;
    Thread t;

    /** Creates a new instance of jreversi */
    public jreversi() {
        AI = new ReversiAI();

        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    gui = new ReversiGUI(AI);
                    AI.setGui(gui);
                }
            });
        } catch (Exception e){System.out.println(e);}

        t = new Thread(this);
        t.start();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new jreversi();
    }

    /*
     * main thread
     */
    public void run() {
        Point move;
        byte row,col,turn;

        gui.setNewGameStarted(false);

        for(;;){
            // wait for new game request
            while (!gui.isNewGameStarted()){
                try {
                    Thread.sleep(T_TEST);
                } catch(InterruptedException ie) {}
            }
            gameFinished = false;
            gui.newGame();
            gui.setNewGameStarted(false);


            // an entire game
            while((!gui.isNewGameStarted())&&(!gameFinished)){
                if (!AI.canMakeMove()){
                    //One is not able to change the turn
                    changeTurn();
                    if (!AI.canMakeMove()){
                        gui.finish(); // end of the game
                        gameFinished = true;
                        break;
                    }
                }

                if ((AI.getTurn() == ReversiAI.WHITE)||(!AI.isSinglePlayer())){
                    while(((move =  AI.getHumanMove()) == null)&&(!gui.isNewGameStarted())){
                        try {
                            Thread.sleep(T_TEST);
                        } catch(InterruptedException ie) {}
                    }
                    AI.setHumanMove(null);

                }else{
```

```
                    move = AI.suggest();
                    try {
                        Thread.sleep(getPauseTime(AI.getPlys()));
                    } catch (InterruptedException ie) {}
                    gui.setProgress(100);
                }

                if(gui.isNewGameStarted()){
                    break;
                }

                row = (byte)move.x;
                col = (byte)move.y;
                if (AI.isValidMove(row,col)) {
                    doMove(row,col,AI.update(row,col));
                    changeTurn();
                }else if ((AI.getTurn() == ReversiAI.BLACK)&&(AI.isSinglePlayer())){
                    // AI cannot find a valid move
                    changeTurn();
                }
            }
        }
    }

    private long getPauseTime(byte plys){
        switch (plys){
            case 1:
                return (long)(1.5*T_PAUSE);
            case 2:
                return (long)(1.5*T_PAUSE);
            case 3:
                return (long)(1.5*T_PAUSE);
            case 4:
                return (T_PAUSE);
            case 5:
                return (long)(T_PAUSE);
            case 6:
                return (long)(0.25*T_PAUSE);
            default:
                return 0;
       }
    }

    private void doMove(int row, int col, LinkedHashSet s){
        final Point p = new Point(row,col);
        final LinkedHashSet takes = s;

        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run(){
                    gui.doMove(p.x,p.y,takes);
                }
            });
        } catch (Exception e){System.out.println(e);}
    }
    private void changeTurn() {
        AI.changeTurn();
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                gui.changeTurn(AI.getTurn());
            }
        });
    }
}
```

Código Fuente de la clase de la IA, ReversiAI:

```
import java.awt.*;
import java.util.*;

public class ReversiAI {
    public static final byte EMPTY = 0;
    public static final byte WHITE = 1;
    public static final byte BLACK = 2;

    private static int [][] weights = {
        {120, -40, 20,  5,  5, 20, -40, 120},
        {-40, -60, -5, -5, -5, -5, -60, -40},
        { 20,  -5, 15,  3,  3, 15,  -5,  20},
        {  5,  -5,  3,  3,  3,  3,  -5,   5},
        {  5,  -5,  3,  3,  3,  3,  -5,   5},
        { 20,  -5, 15,  3,  3, 15,  -5,  20},
        {-40, -60, -5, -5, -5, -5, -60, -40},
        {120, -40, 20,  5,  5, 20, -40, 120}
    };

    private byte plys;
```

```
private byte twidth;
private byte theight;
private byte [][] currentBoard;
private byte turn;
private Point humanMove;
private boolean singlePlayer = true;
private ReversiGUI gui;

/** Creates a new instance of ReversiAI
 *  By default, a 8x8 board and the most simple
 *  depth for the algorithm is used.
 */
public ReversiAI() {
    ReversiAIbuilder((byte)8, (byte)8, (byte)-1);
}


/** Creates a new instance of ReversiAI with
 *  a heightxwidth board and p depth for the
 *  algorithm.
 */
public void ReversiAIbuilder(byte height, byte width, byte p) {
    plys = p;
    twidth = width;
    theight = height;
    turn = ReversiAI.WHITE;

    currentBoard = new byte[twidth][theight];
    for (int i = 0; i < twidth; i++)
        for (int j = 0; j < theight; j++)
            currentBoard[i][j] = ReversiAI.EMPTY;

    currentBoard[3][3] = ReversiAI.WHITE;
    currentBoard[3][4] = ReversiAI.BLACK;
    currentBoard[4][3] = ReversiAI.BLACK;
    currentBoard[4][4] = ReversiAI.WHITE;

}

/** Returns a table with the current game status
 */
public byte[][] getCurrentStatus() {
    return currentBoard;
}


/** Returns board height
 */
public int getHeight() {
    return this.theight;
}

/** Returns board width
 */
public int getWidth() {
    return this.twidth;
}

/** Returns the "plys" used in minimax algorithm by the AI
 */
public byte getPlys() {
    return this.plys;
}

public void setHumanMove(Point p){
    this.humanMove = p;
}

public Point getHumanMove(){
    return this.humanMove;
}

/** Returns the current turn
 */
public byte getTurn() {
    return this.turn;
}

/** Changes player's turn to the next one
 */
public void changeTurn() {
    if (this.turn == ReversiAI.WHITE)
        this.turn = ReversiAI.BLACK;
    else
        this.turn = ReversiAI.WHITE;
}

/** Set the "plys" to use by the AI
```

```
 */
public void setPlys(byte p) {
    this.plys = p;
}

public boolean isSinglePlayer(){
    return this.singlePlayer;
}

public void setSinglePlayer(boolean p) {
    this.singlePlayer = p;
}

public void setGui(ReversiGUI g) {
    this.gui = g;
}

/*Dado un estado (tablero,turno) la función EXPANDIR
  nos va a calcular las posibles jugadas legales
  a partir del mismo.
 */
private ArrayList expandir(byte [][] board,byte turn) {
    ArrayList moves = new ArrayList();
    for (byte i = 0; i < theight; i++){
        for (byte j = 0; j < twidth; j++){
            if (board[i][j]==ReversiAI.EMPTY){
                //isValid hace un recorrido, abortando en cuanto encuentre
                //un recorrido válido devuelve TRUE con lo cual
                //podemos meter un movimiento válido en la lista
                //con poco coste de computación, los volteos efectivo para llegar al
                //nodo desde el que se expande se realizan antes de llamar a expandir
                if (isValidTableMove(board,turn,i,j)){
                    moves.add(new Point(i,j));
                }
            }
        }
    }
    return moves;
}

public boolean isValidMove(byte row, byte col) {
    return isValidTableMove(currentBoard, turn, row,col);
}

private boolean isValidTableMove(byte [][] board, byte turn, byte row, byte col) {
    LinkedHashSet s = new LinkedHashSet();
    if (board[row][col] == ReversiAI.EMPTY) {  // only empty positions can be used!
        for (byte y = (byte)((col > 0) ? col - 1 : col);
        y <= ((col < (twidth-1)) ? col + 1 : col); y++) {
            for (byte x = (byte)((row>0) ? row - 1 : row);
            x <= ((row < (theight - 1)) ? row + 1 : row); x++) {
                if ((board[x][y] != turn) && (board[x][y] != ReversiAI.EMPTY)) {
                    if (linealSearch(board,row,col,x,y,turn,s))
                        return true;
                }
            }
        }
    }
    return false;
}

private LinkedHashSet getTakes(byte [][] board, byte turn, byte row, byte col) {

    LinkedHashSet s = new LinkedHashSet();

    if (board[row][col] == ReversiAI.EMPTY) {  // only empty positions can be used!
        for (byte y = (byte)((col > 0) ? col - 1 : col);
        y <= ((col < (twidth-1)) ? col + 1 : col); y++) {
            for (byte x = (byte)((row>0) ? row - 1 : row);
            x <= ((row < (theight - 1)) ? row + 1 : row); x++) {
                if ((board[x][y] != turn) && (board[x][y] != ReversiAI.EMPTY)) {
                    linealSearch(board,row,col,x,y,turn,s);
                }

            }
        }
    }
    return s;

}

private boolean linealSearch(byte[][] board,byte ox,byte oy,
                             byte dx,byte dy, byte t,LinkedHashSet s) {
    LinkedHashSet ns = new LinkedHashSet();
    int vx = dx - ox;
    int vy = dy - oy;
    int i = 0;
    Point p;
```

```
        dy = (byte)(dy + vy);
        dx = (byte)(dx + vx);
        while ((dx >= 0) && (dy >= 0) &&
                (dx <= (theight -1)) &&
                (dy <= (twidth -1))) {

            if (board[dx][dy] == t) {
                p = new Point(dx - vx,dy - vy);
                ns.add(p);
                s.addAll(ns);
                return true;
            }

            if (board[dx][dy] == ReversiAI.EMPTY)
                return false;

            // !t && !empty
            p = new Point(dx - vx,dy - vy);
            ns.add(p);


            dy = (byte)(dy + vy);
            dx = (byte)(dx + vx);
        }
        return false;
    }


    /** returns whether or not a player could make at least one legal
     *  move
     */
    public boolean canMakeMove() {
        return turnCanMakeMove(currentBoard,turn);

    }

    public boolean turnCanMakeMove(byte [][] board, byte turn) {
        for (byte i = 0; i < theight; i++){
            for (byte j = 0; j < twidth; j++){
                if (board[i][j]==ReversiAI.EMPTY){
                    //isValid hace un recorrido, abortando en cuanto encuentre
                    //un recorrido válido devuelve TRUE con lo cual
                    //podemos meter un movimiento válido en la lista
                    //con poco coste de computación, los volteos efectivo para llegar al
                    //nodo desde el que se expande se realizan antes de llamar a expandir
                    if (isValidTableMove(board,turn,i,j)){
                        return true;
                    }
                }
            }
        }
        return false;
    }

    /** Update the game tree to reflect the current status
     *  when a move was done in (x,y).
     */
    public LinkedHashSet update(byte x, byte y) {
        LinkedHashSet s = getTakes(currentBoard,turn, x,y);
        currentBoard = doMove(currentBoard,turn, new Point(x, y), s);
        return s;
    }

    /** Suggest the best possible move given the current game status
     */
    public Point suggest() {
        int sc ,bestSc = Integer.MIN_VALUE;
        ArrayList l;
        Iterator i;
        Point move,best;
        best = new Point(-1, -1);
        l = expandir(currentBoard, ReversiAI.BLACK);
        if (l.isEmpty()){
            return new Point(-1,-1);
            //This point must never be reached because we are sure
            //AI can make a move, so "L" will never be empty
        }
        i = l.iterator();
        while (i.hasNext()) {
            move = (Point)(i.next());
            LinkedHashSet takes = getTakes(currentBoard, ReversiAI.BLACK, (byte)move.x,(byte)move.y);
            byte [][] board = doMove(currentBoard, ReversiAI.BLACK, move,takes);
            sc = minimax(board,ReversiAI.BLACK, 1, Integer.MIN_VALUE,Integer.MAX_VALUE);
            if (sc>=bestSc){
                bestSc = sc;
                best = move;
            }
```

```
        }
        return best;
    }


    private int max(int a, int b){
        if (a>b)
            return a;
        return b;
    }

    private int min(int a, int b){
        if (a<b)
            return a;
        return b;
    }

    public byte[][] cloneTable(byte[][] table) {
        byte[][] t = new byte[twidth][theight];

        for (byte x = 0; x < theight; x++)
            for (byte y = 0; y < twidth; y++)
                t[x][y] = table[x][y];

        return t;
    }


    private byte[][] doMove(byte[][] board,byte turn,Point move,LinkedHashSet takes){
        byte x,y;
        byte [][] t = cloneTable(board);

        if (turn == ReversiAI.WHITE) {
            t[move.x][move.y] = ReversiAI.WHITE;
        } else {
            t[move.x][move.y] = ReversiAI.BLACK;
        }
        Iterator it = takes.iterator();
        while (it.hasNext()) {
            Point p = (Point)it.next();
            x = (byte)p.getX();
            y = (byte)p.getY();
            if (t[x][y] == ReversiAI.WHITE)
                t[x][y] = ReversiAI.BLACK;
            else
                t[x][y] = ReversiAI.WHITE;
        }
        return t;
    }

    public byte invertTurn(byte turn) {
        if (turn == ReversiAI.WHITE)
            return  ReversiAI.BLACK;
        else
            return  ReversiAI.WHITE;
    }

    private int minimax(byte [][] board,byte turn, int depth,int alpha, int beta) {
        byte currentTurn = turn;
        ArrayList sons = new ArrayList();
        LinkedHashSet s;
        Iterator i;
        Point move;
        int v, score;
        int ex =0;
        byte [][] newBoard;

        if (depth>=plys)
            return e(board,false);

        // Itentamos expandir para el turno que toca
        // si no podemos intentamos para el otro turno llamando
        // a la función contraria para intentar haces dos movimientos
        // seguidos
        sons = expandir(board, currentTurn);
        //Si no pude expandir intento con el otro turno
        if (sons.isEmpty()){
            currentTurn = invertTurn(currentTurn);
            sons = expandir(board, currentTurn);
        }
        if (sons.isEmpty()){
            //No one can move, the game has ended
            if (gui.getProgress() < 100/(depth+1))
                gui.setProgress(100/(depth+1));
            return e(board,true);
        }
        i = sons.iterator();
        while (i.hasNext()) {
```

```
            move =(Point)(i.next());
            s = getTakes(board, currentTurn, (byte)move.x,(byte)move.y);
            newBoard = doMove(board, currentTurn, move, s);

            ex = minimax(newBoard, invertTurn(currentTurn), depth+1, alpha, beta);
            if (currentTurn == ReversiAI.BLACK){
                //This is a MAX node so we must take the highest value
                //fron the child we take a child
                if (ex > alpha)
                    alpha = ex;
                if (alpha >= beta){
                    return beta;
                }
            } else {
                //Poda
                if (ex < beta){
                    beta = ex;
                }
                if (beta<=alpha)
                    return alpha;
            }
        }

        if (gui.getProgress() < 100/(depth+1))
            gui.setProgress(100/(depth+1));

        if (currentTurn == ReversiAI.BLACK)
            return alpha;
        else
            return beta;

    }

    public boolean isFinal(byte[][] board,byte cntwhite,byte cntblack,boolean F){
        byte cntfichas = 0;
        ArrayList sons = new ArrayList();
        if (F)
            return true;
        if ((cntwhite + cntblack) == 64)
            return true;
        if ((!turnCanMakeMove(board, ReversiAI.BLACK))
        &&(!turnCanMakeMove(board, ReversiAI.WHITE))){
            return true;
        }
        return false;
    }

    /* Evaluation function for a board and a player.
     * It uses the global "turn" to determine who
     * is the evaluation for. The evaluation cosists
     * in adding the player pieces weights and subtracts
     * the opponent's
     */
    private int e(byte [][] board,boolean F){
        byte cntwhite= 0,cntblack = 0;
        int count = 0;
        int w;

        for (byte i = 0; i < theight; i++){
            for(byte j = 0; j < twidth; j++){
                if (board[i][j]!=ReversiAI.EMPTY){
                    if ((((  i == 0 && j == 1 ) ||
                            ( i == 1 && j == 0 ) ||
                            ( i == 1 && j == 1 )) &&
                            board[0][0]!= ReversiAI.EMPTY ) ||
                            ((( i == 0 && j == 6 ) ||
                            ( i == 1 && j == 6 ) ||
                            ( i == 1 && j == 7 )) &&
                            board[0][7] != ReversiAI.EMPTY ) ||
                            ((( i == 6 && j == 0 ) ||
                            ( i == 7 && j == 1 ) ||
                            ( i == 7 && j == 1 )) &&
                            board[7][0] != ReversiAI.EMPTY ) ||
                            ((( i == 6 && j == 7 ) ||
                            ( i == 6 && j == 6 ) ||
                            ( i == 7 && j == 6 )) &&
                            board[7][7] != ReversiAI.EMPTY ) )
                        w = ( 5 - weights[i][j] ) * ((board[i][j] == ReversiAI.BLACK)?1:-1);
                    else
                        w = weights[i][j];
                    if (board[i][j] == ReversiAI.BLACK){
                        count +=w;
                        cntblack++;
                    } else if (board[i][j] == ReversiAI.WHITE){
                        count -=w;
                        cntwhite++;
                    }
                }
```

```
            }
        }
        if (isFinal(board, cntwhite,cntblack,F)){
            //The AI detected a final situation in
            //possible in the future
            if (cntblack>cntwhite)
                return Integer.MAX_VALUE;
            else if (cntblack<cntwhite)
                return Integer.MIN_VALUE;
            else
                return 0;
        }
        return count;
    }
}
```

Código Fuente de la clase de control de la GUI, ReversiGUI:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.text.*;

public class ReversiGUI extends javax.swing.JFrame {

    private static final byte MAX_LEVEL = 12;

    /** The table model is used to represent data in the table.
     *  This class extends the functionality of the default one by
     *  substituting the objects in the cells by images which can be
     *  changed in runtime.
     */
    class ReversiTableModel extends javax.swing.table.AbstractTableModel {
        // basic states
        private ImageIcon empty = new ImageIcon("img/empty.gif","empty");
        private ImageIcon black = new ImageIcon("img/black.gif","black");
        private ImageIcon white = new ImageIcon("img/white.gif","white");
        // for common animations
        private ImageIcon unscaled_w2b = new ImageIcon("img/w2b.gif","black");
        private ImageIcon unscaled_b2w = new ImageIcon("img/b2w.gif", "white");
        private ImageIcon w2b;
        private ImageIcon b2w;

        private String[] columnNames = {"","","","","","","","",""};
        private Object[][] data = {
            {empty,empty,empty,empty,empty,empty,empty,empty},
            {empty,empty,empty,empty,empty,empty,empty,empty},
            {empty,empty,empty,empty,empty,empty,empty,empty},
            {empty,empty,empty,white,black,empty,empty,empty},
            {empty,empty,empty,black,white,empty,empty,empty},
            {empty,empty,empty,empty,empty,empty,empty,empty},
            {empty,empty,empty,empty,empty,empty,empty,empty},
            {empty,empty,empty,empty,empty,empty,empty,empty}
        };

        public int getColumnCount() {
            return data.length;
        }

        public int getRowCount() {
            return data.length;
        }

        public String getColumnName(int col) {
            return columnNames[col];
        }

        public Object getValueAt(int row, int col) {
            return data[row][col];
        }

        public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass();
        }

        public boolean isCellEditable(int row, int col) {
            return false;
        }

        public void setValueAt(Object value, int row, int col) {
            data[row][col] = value;
            fireTableCellUpdated(row,col);
        }
```

```
    public void setWhite(int row, int col) {
        int width = getWidth() / data.length /2;
        if (((ImageIcon)data[row][col]).getDescription() == "empty") {
            setValueAt(white,row,col);
        } else if (((ImageIcon)data[row][col]).getDescription() == "black") {
            b2w = new ImageIcon(unscaled_b2w.getImage().getScaledInstance(width, width,
                    Image.SCALE_FAST));
            b2w.setImageObserver(iobs);
            b2w.setDescription("white");
            setValueAt(b2w,row,col);
        }
    }

    public void setBlack(int row, int col) {
        int width = getWidth() / data.length /2;
        if (((ImageIcon)data[row][col]).getDescription() == "empty") {
            setValueAt(black,row,col);
        } else if (((ImageIcon)data[row][col]).getDescription() == "white") {
            w2b = new ImageIcon(unscaled_w2b.getImage().getScaledInstance(width, width,
                    Image.SCALE_FAST));
            w2b.setImageObserver(iobs);
            w2b.setDescription("black");
            setValueAt(w2b,row,col);
        }
    }

    public void scaleImages() {
        int width = getWidth() / data.length /2;
        empty.setImage(empty.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        black.setImage(black.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        white.setImage(white.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        b2w = new ImageIcon(unscaled_b2w.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        b2w.setImageObserver(iobs);
        w2b = new ImageIcon(unscaled_w2b.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        w2b.setImageObserver(iobs);
    }

    public boolean isEmpty(int row, int col) {
        return ((ImageIcon)data[row][col]).getDescription() == "empty";
    }

    public boolean isBlack(int row, int col) {
        return ((ImageIcon)data[row][col]).getDescription() == "black";
    }

    public boolean isWhite(int row, int col) {
        return ((ImageIcon)data[row][col]).getDescription() == "white";
    }

    public void reset() {
        for (int i = 0; i < getRowCount(); i++)
            for (int j = 0; j < getColumnCount(); j++)
                data[i][j] = empty;
        data[3][3] = white;
        data[3][4] = black;
        data[4][3] = black;
        data[4][4] = white;

    }

    public ImageObserver iobs = new ImageObserver() {
        public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
            repaint();
            return true;
        }};
};

/** Creates new form ReversiGUI */
public ReversiGUI(ReversiAI rai) {
    ReversiTableModel model = new ReversiTableModel();
    AI = rai;

    initComponents();
    this.setLocationRelativeTo(null);
    setVisible(true);
    timeListener = new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            /***************************** This is a dirty hack!
             * Date and Time methods seem to need negative values
             * to represent the first hour, so an effective value
             * of 1000 miliseconds translates into an hour and 1
             * second, wherever a single second was expected.
```

```
             */
            java.sql.Time time = new java.sql.Time(t -1000*60*60);
            t += 1000;
            TimeLabel.setText(time.toString());
        }
    };
    GameTimer = new javax.swing.Timer(1000,timeListener);
    PlayerDialog.setSize(200,160);

    model.scaleImages();
    Board.setModel(model);
    Board.setRowHeight(Board.getWidth() / Board.getColumnModel().getColumnCount());
    initBoard();
}

/** This method is called at the begining and when user requests
 *  a new game to reset and initialize the board.
 */
private void initBoard() {
    ReversiTableModel model;

    model = (ReversiTableModel)Board.getModel();

    /* Initialize board */
    model.reset();
    Board.setEnabled(false);
    ComputerScore.setText("0");
    PlayerScore.setText("0");
}

private byte readLevel(){

    // level selection
    try {
        Object[] levels = new Object[MAX_LEVEL];
        for (byte i = 0; i < MAX_LEVEL; i++)
            levels[i] = (new Integer(i+1)).toString();
        String s = (String)(JOptionPane.showInputDialog(null,
                "Select your preferred dificulty level:",
                "Dificulty level",
                JOptionPane.PLAIN_MESSAGE,
                null,levels,"5"));
        if (s == null)
            return -1;
        return  (byte)(Integer.parseInt(s));
    } catch (NumberFormatException e){ return (byte)5; }
}

public void newGame(){
    byte plys = 1;
    boolean singlePlayer = true;

    initBoard();

    // number of players selection
    Object[] numplayers = {"Single player", "Two players"};
    String s =(String)JOptionPane.showInputDialog(null,
            "Select the game mode you want to play:",
            "Game mode",
            JOptionPane.PLAIN_MESSAGE,
            null,numplayers,"Single player");
    if (s == "Two players") {
        singlePlayer = false;
        jLabel1.setText("Player 2");
        jComboBox1.setEnabled(true);
        if (jLabel2.getText() == "Player")
            jLabel2.setText("Player 1");
    } else if (s == "Single player") {
        singlePlayer = true;
        jLabel1.setText("Computer");
        jComboBox1.setEnabled(false);
    } else return;

    AI.ReversiAIbuilder((byte)8,(byte)8,plys);
    AI.setSinglePlayer(singlePlayer);

    if (singlePlayer) {
        plys = readLevel();
        if (plys == -1)
            return;
        jLabel3.setText("Level "+ plys);
    }

    Board.setEnabled(true);

    ComputerScore.setText("2");
    PlayerScore.setText("2");
```

```
        //timer
        t = 1000;
        GameTimer.restart();

        // paint turn colours
        changeTurn(ReversiAI.WHITE);


    }

    public void setNewGameStarted(boolean v){
        newgamestarted = v;
    }

    public boolean isNewGameStarted(){
        return newgamestarted;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        PlayerDialog = new javax.swing.JDialog();
        AcceptNameButton = new javax.swing.JButton();
        jLabel23 = new javax.swing.JLabel();
        playerName = new javax.swing.JTextField();
        jComboBox1 = new javax.swing.JComboBox();
        jPanel1 = new javax.swing.JPanel();
        Board = new javax.swing.JTable();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        jLabel12 = new javax.swing.JLabel();
        jLabel13 = new javax.swing.JLabel();
        jLabel14 = new javax.swing.JLabel();
        jLabel15 = new javax.swing.JLabel();
        jLabel16 = new javax.swing.JLabel();
        jLabel17 = new javax.swing.JLabel();
        jLabel18 = new javax.swing.JLabel();
        jLabel19 = new javax.swing.JLabel();
        jLabel20 = new javax.swing.JLabel();
        jLabel21 = new javax.swing.JLabel();
        jLabel22 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jPanel3 = new javax.swing.JPanel();
        ComputerScore = new javax.swing.JLabel();
        jPanel4 = new javax.swing.JPanel();
        PlayerScore = new javax.swing.JLabel();
        TimeLabel = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jSeparator1 = new javax.swing.JSeparator();
        jProgressBar1 = new javax.swing.JProgressBar();
        jMenuBar1 = new javax.swing.JMenuBar();
        GameMenu = new javax.swing.JMenu();
        MenuNew = new javax.swing.JMenuItem();
        MenuPlayer = new javax.swing.JMenuItem();
        jSeparator2 = new javax.swing.JSeparator();
        MenuExit = new javax.swing.JMenuItem();
        HelpMenu = new javax.swing.JMenu();
        MenuAbout = new javax.swing.JMenuItem();

        PlayerDialog.getContentPane().setLayout(new java.awt.GridBagLayout());

        PlayerDialog.setTitle("Change name");
        PlayerDialog.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        PlayerDialog.setResizable(false);
        AcceptNameButton.setText("Accept");
        AcceptNameButton.setMinimumSize(new java.awt.Dimension(50, 10));
        AcceptNameButton.setPreferredSize(new java.awt.Dimension(50, 10));
        AcceptNameButton.setRequestFocusEnabled(false);
        AcceptNameButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AcceptNameButtonActionPerformed(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
```

```
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 50;
gridBagConstraints.ipady = 15;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(0, 30, 20, 30);
PlayerDialog.getContentPane().add(AcceptNameButton, gridBagConstraints);

jLabel23.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel23.setText("Please, enter your name:");
jLabel23.setMinimumSize(new java.awt.Dimension(100, 10));
jLabel23.setPreferredSize(new java.awt.Dimension(100, 10));
jLabel23.setRequestFocusEnabled(false);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 70;
gridBagConstraints.ipady = 10;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(20, 10, 10, 10);
PlayerDialog.getContentPane().add(jLabel23, gridBagConstraints);

playerName.setText(jLabel2.getText());
playerName.setMinimumSize(new java.awt.Dimension(100, 10));
playerName.setPreferredSize(new java.awt.Dimension(100, 10));
playerName.setRequestFocusEnabled(false);
playerName.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        playerNameActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 70;
gridBagConstraints.ipady = 10;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(0, 10, 10, 10);
PlayerDialog.getContentPane().add(playerName, gridBagConstraints);

jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[]
         { "Player 1", "Player 2" }));
jComboBox1.setEnabled(false);
jComboBox1.setRequestFocusEnabled(false);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 70;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(0, 10, 10, 10);
PlayerDialog.getContentPane().add(jComboBox1, gridBagConstraints);

getContentPane().setLayout(new java.awt.GridBagLayout());

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("jReversi");
setName("MainFrame");
setResizable(false);
jPanel1.setLayout(null);

jPanel1.setBorder(new javax.swing.border.TitledBorder("Board"));
jPanel1.setName("BoardPanel");
Board.setBackground(new java.awt.Color(0, 92, 0));
Board.setBorder(new javax.swing.border.BevelBorder(javax.swing.border.BevelBorder.LOWERED));
Board.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null},
        {null, null, null, null, null, null, null, null}
    },
    new String [] {
       "Title 1", "Title 2", "Title 3", "Title 4", "Title 5", "Title 6", "Title 7", "Title 8"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false, false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
```

```
        return canEdit [columnIndex];
    }
});
Board.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
Board.setAutoscrolls(false);
Board.setDoubleBuffered(true);
Board.setEnabled(false);
Board.setGridColor(new java.awt.Color(0, 51, 0));
Board.setIntercellSpacing(new java.awt.Dimension(0, 0));
Board.setMaximumSize(new java.awt.Dimension(400, 400));
Board.setMinimumSize(new java.awt.Dimension(400, 400));
Board.setName("Panel");
Board.setPreferredSize(new java.awt.Dimension(250, 250));
Board.setRowSelectionAllowed(false);
Board.setSelectionBackground(new java.awt.Color(255, 255, 255));
Board.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        BoardMouseClicked(evt);
    }
});

jPanel1.add(Board);
Board.setBounds(20, 30, 250, 250);

jLabel5.setFont(new java.awt.Font("Courier", 1, 10));
jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel5.setText("A    B    C    D    E    F    G    H");
jLabel5.setFocusable(false);
jLabel5.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jPanel1.add(jLabel5);
jLabel5.setBounds(20, 10, 250, 20);

jLabel6.setFont(new java.awt.Font("Courier", 1, 10));
jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel6.setText("A    B    C    D    E    F    G    H");
jLabel6.setFocusable(false);
jLabel6.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jPanel1.add(jLabel6);
jLabel6.setBounds(20, 280, 250, 20);

jLabel7.setFont(new java.awt.Font("Courier", 1, 10));
jLabel7.setText("1");
jPanel1.add(jLabel7);
jLabel7.setBounds(275, 35, 10, 20);

jLabel8.setFont(new java.awt.Font("Courier", 1, 10));
jLabel8.setText("2");
jPanel1.add(jLabel8);
jLabel8.setBounds(275, 66, 10, 20);

jLabel9.setFont(new java.awt.Font("Courier", 1, 10));
jLabel9.setText("3");
jPanel1.add(jLabel9);
jLabel9.setBounds(275, 97, 10, 20);

jLabel10.setFont(new java.awt.Font("Courier", 1, 10));
jLabel10.setText("4");
jPanel1.add(jLabel10);
jLabel10.setBounds(275, 128, 10, 20);

jLabel11.setFont(new java.awt.Font("Courier", 1, 10));
jLabel11.setText("5");
jPanel1.add(jLabel11);
jLabel11.setBounds(275, 159, 10, 20);

jLabel12.setFont(new java.awt.Font("Courier", 1, 10));
jLabel12.setText("6");
jPanel1.add(jLabel12);
jLabel12.setBounds(275, 190, 10, 20);

jLabel13.setFont(new java.awt.Font("Courier", 1, 10));
jLabel13.setText("7");
jPanel1.add(jLabel13);
jLabel13.setBounds(275, 221, 10, 20);

jLabel14.setFont(new java.awt.Font("Courier", 1, 10));
jLabel14.setText("8");
jPanel1.add(jLabel14);
jLabel14.setBounds(275, 252, 10, 20);

jLabel15.setFont(new java.awt.Font("Courier", 1, 10));
jLabel15.setText("8");
jPanel1.add(jLabel15);
jLabel15.setBounds(9, 252, 10, 20);

jLabel16.setFont(new java.awt.Font("Courier", 1, 10));
jLabel16.setText("7");
jPanel1.add(jLabel16);
```

```
jLabel16.setBounds(9, 221, 10, 20);

jLabel17.setFont(new java.awt.Font("Courier", 1, 10));
jLabel17.setText("6");
jPanel1.add(jLabel17);
jLabel17.setBounds(9, 190, 10, 20);

jLabel18.setFont(new java.awt.Font("Courier", 1, 10));
jLabel18.setText("5");
jPanel1.add(jLabel18);
jLabel18.setBounds(9, 159, 10, 20);

jLabel19.setFont(new java.awt.Font("Courier", 1, 10));
jLabel19.setText("4");
jPanel1.add(jLabel19);
jLabel19.setBounds(9, 128, 10, 20);

jLabel20.setFont(new java.awt.Font("Courier", 1, 10));
jLabel20.setText("3");
jPanel1.add(jLabel20);
jLabel20.setBounds(9, 97, 10, 20);

jLabel21.setFont(new java.awt.Font("Courier", 1, 10));
jLabel21.setText("2");
jPanel1.add(jLabel21);
jLabel21.setBounds(9, 66, 10, 20);

jLabel22.setFont(new java.awt.Font("Courier", 1, 10));
jLabel22.setText("1");
jPanel1.add(jLabel22);
jLabel22.setBounds(9, 35, 10, 20);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 289;
gridBagConstraints.ipady = 299;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jPanel1, gridBagConstraints);

jPanel2.setLayout(null);

jPanel2.setBorder(new javax.swing.border.TitledBorder("Score"));
jPanel2.setName("ScorePanel");
jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel1.setText("Computer");
jPanel2.add(jLabel1);
jLabel1.setBounds(10, 20, 130, 20);

jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel2.setText("Player");
jPanel2.add(jLabel2);
jLabel2.setBounds(10, 90, 130, 20);

jPanel3.setLayout(null);

jPanel3.setBackground(new java.awt.Color(0, 0, 0));
jPanel3.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(0, 0, 0), 1, true));
ComputerScore.setBackground(new java.awt.Color(0, 0, 0));
ComputerScore.setFont(new java.awt.Font("Courier", 0, 14));
ComputerScore.setForeground(new java.awt.Color(255, 255, 255));
ComputerScore.setText("0");
jPanel3.add(ComputerScore);
ComputerScore.setBounds(4, 6, 30, 20);

jPanel2.add(jPanel3);
jPanel3.setBounds(55, 50, 40, 30);

jPanel4.setLayout(null);

jPanel4.setBackground(new java.awt.Color(255, 255, 255));
jPanel4.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(0, 0, 0), 1, true));
PlayerScore.setFont(new java.awt.Font("Courier", 0, 14));
PlayerScore.setText("0");
jPanel4.add(PlayerScore);
PlayerScore.setBounds(4, 6, 30, 20);

jPanel2.add(jPanel4);
jPanel4.setBounds(55, 120, 40, 30);

TimeLabel.setFont(new java.awt.Font("Courier", 1, 14));
TimeLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
TimeLabel.setText("00:00:00");
jPanel2.add(TimeLabel);
TimeLabel.setBounds(5, 274, 140, 20);

jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
```

```
jPanel2.add(jLabel3);
jLabel3.setBounds(10, 220, 130, 20);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipady = 100;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jPanel2, gridBagConstraints);

jSeparator1.setBorder(new javax.swing.border.BevelBorder
                      (javax.swing.border.BevelBorder.RAISED));
jSeparator1.setName("Separator");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 289;
gridBagConstraints.ipady = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jSeparator1, gridBagConstraints);

jProgressBar1.setName("ProgressBar");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 289;
gridBagConstraints.ipady = 6;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(4, 0, 2, 0);
getContentPane().add(jProgressBar1, gridBagConstraints);

GameMenu.setMnemonic('G');
GameMenu.setText("Game");
GameMenu.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        GameMenuActionPerformed(evt);
    }
});

MenuNew.setMnemonic('N');
MenuNew.setText("New Game");
MenuNew.setName("MenuNew");
MenuNew.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        MenuNewActionPerformed(evt);
    }
});

GameMenu.add(MenuNew);

MenuPlayer.setMnemonic('P');
MenuPlayer.setText("Player name");
MenuPlayer.setName("MenuPlayer");
MenuPlayer.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        MenuPlayerActionPerformed(evt);
    }
});

GameMenu.add(MenuPlayer);

GameMenu.add(jSeparator2);

MenuExit.setMnemonic('Q');
MenuExit.setText("Quit");
MenuExit.setName("MenuExit");
MenuExit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        MenuExitActionPerformed(evt);
    }
});

GameMenu.add(MenuExit);

jMenuBar1.add(GameMenu);

HelpMenu.setMnemonic('H');
HelpMenu.setText("Help");
MenuAbout.setMnemonic('A');
MenuAbout.setText("About");
MenuAbout.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        HelpAbout(evt);
```

```
        }
    });

    HelpMenu.add(MenuAbout);

    jMenuBar1.add(HelpMenu);

    setJMenuBar(jMenuBar1);

    pack();
}

private void HelpAbout(java.awt.event.ActionEvent evt) {
    JOptionPane.showMessageDialog(null,
            "jReversi developers are:\n" +
            "* Tomás Aguado Gómez\n" +
            "* Angel Jara Gómez\n" +
            "* Jaime Pérez Crespo",
            "About jReversi", JOptionPane.INFORMATION_MESSAGE);
}


private void MenuNewActionPerformed(java.awt.event.ActionEvent evt) {
    newgamestarted = true;
}

private void playerNameActionPerformed(java.awt.event.ActionEvent evt) {
    if (playerName.getText().length() != 0) {
        if (jComboBox1.getSelectedItem() == "Player 1")
            jLabel2.setText(playerName.getText());
        else
            jLabel1.setText(playerName.getText());

        PlayerDialog.hide();
    }
}

private void AcceptNameButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (playerName.getText().length() != 0) {
        if (jComboBox1.getSelectedItem() == "Player 1")
            jLabel2.setText(playerName.getText());
        else
            jLabel1.setText(playerName.getText());

        PlayerDialog.hide();
    }
}

private void MenuPlayerActionPerformed(java.awt.event.ActionEvent evt) {
    playerName.setText(jLabel2.getText());
    PlayerDialog.setLocationRelativeTo(this);
    PlayerDialog.show();
}

private void MenuExitActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

public void doMove(int row, int col,LinkedHashSet s) {
    ReversiTableModel m = (ReversiTableModel)Board.getModel();
    int score;
    int t = AI.getTurn();

    if (t == ReversiAI.WHITE) {
        score = Integer.parseInt(PlayerScore.getText());
        PlayerScore.setText(Integer.toString(++score));
        m.setWhite(row,col);
    } else {
        score = Integer.parseInt(ComputerScore.getText());
        ComputerScore.setText(Integer.toString(++score));
        m.setBlack(row,col);
    }
    Iterator i = s.iterator();
    while (i.hasNext()) {
        Point p = (Point)i.next();
        if (m.isWhite((int)p.getX(),(int)p.getY())) {
            score = Integer.parseInt(ComputerScore.getText());
            ComputerScore.setText(Integer.toString(++score));
            score = Integer.parseInt(PlayerScore.getText());
            PlayerScore.setText(Integer.toString(--score));
            m.setBlack((int)p.getX(),(int)p.getY());
        } else {
            score = Integer.parseInt(PlayerScore.getText());
            PlayerScore.setText(Integer.toString(++score));
            score = Integer.parseInt(ComputerScore.getText());
            ComputerScore.setText(Integer.toString(--score));
            m.setWhite((int)p.getX(),(int)p.getY());
        }
```

```
        }
        setToolTip(row, col);
    }

    private void setToolTip(int col, int row) {
        char c = 'A';

        c = (char)((int) c + col);
        Board.setToolTipText("Last move: "
                +Character.toString(c)
                +"/"+String.valueOf(row+1));
    }

    public void setProgress(int n) {
        jProgressBar1.setValue(n);
    }

    public int getProgress() {
        return jProgressBar1.getValue();
    }

    public void changeTurn(int turn) {
        // change the current turn in the GUI
        if (turn == ReversiAI.WHITE) {
            jLabel2.setForeground(Color.RED);
            jLabel1.setForeground(Color.BLACK);
        } else {
            jLabel1.setForeground(Color.RED);
            jLabel2.setForeground(Color.BLACK);
        }
    }

    public void finish() {
        int black = Integer.parseInt(ComputerScore.getText());
        int white = Integer.parseInt(PlayerScore.getText());

        GameTimer.stop();
        jLabel1.setForeground(Color.BLACK);
        jLabel2.setForeground(Color.BLACK);
        Board.setEnabled(false);
        if (black > white)
            JOptionPane.showMessageDialog(null,"Game over, "+
                    jLabel1.getText()+" wins!","Game over", JOptionPane.YES_OPTION);
        else
            JOptionPane.showMessageDialog(null,"Game over, "+
                    jLabel2.getText()+" wins!","Game over", JOptionPane.YES_OPTION);
    }

    private void BoardMouseClicked(java.awt.event.MouseEvent evt) {
        int col = Board.getSelectedColumn();
        int row = Board.getSelectedRow();
        Point p;
        int t;

        // board active only if in human player's turn
        if ((Board.isEnabled())&&((AI.getTurn() == ReversiAI.WHITE)||(!AI.isSinglePlayer()))) {
            AI.setHumanMove(new Point(row,col));
        }
    }

    // Variables declaration - do not modify
    private javax.swing.JButton AcceptNameButton;
    private javax.swing.JTable Board;
    private javax.swing.JLabel ComputerScore;
    private javax.swing.JMenu GameMenu;
    private javax.swing.JMenu HelpMenu;
    private javax.swing.JMenuItem MenuAbout;
    private javax.swing.JMenuItem MenuExit;
    private javax.swing.JMenuItem MenuNew;
    private javax.swing.JMenuItem MenuPlayer;
    private javax.swing.JDialog PlayerDialog;
    private javax.swing.JLabel PlayerScore;
    private javax.swing.JLabel TimeLabel;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
```

```
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JProgressBar jProgressBar1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JTextField playerName;
    private javax.swing.Timer GameTimer;
    private long t;
    private ActionListener timeListener;
    private boolean singlePlayer = true;
    private boolean newgamestarted = false;
    private ReversiAI AI;
    // End of variables declaration
}
```

Código fuente (versión 2, con reutilización de cálculo)

Código Fuente de la clase principal Jreversi:

```
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class jreversi implements Runnable {

    public static final long T_TEST = 100;
    public static final long T_PAUSE = 1500;
    private boolean gameFinished = false;
    private ReversiAI AI;
    private ReversiGUI gui;
    Thread t;

    /** Creates a new instance of jreversi */
    public jreversi() {
        AI = new ReversiAI();

        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    gui = new ReversiGUI(AI);
                    AI.setGui(gui);               }
            });
        } catch (Exception e){System.out.println(e);}

        t = new Thread(this);
        t.start();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new jreversi();
    }

    /*
     * main thread
     */
    public void run() {
        Point move;
        byte row,col,turn;

        gui.setNewGameStarted(false);

        for(;;){
            // wait for new game request

            while (!gui.isNewGameStarted()){
                try {
                    Thread.sleep(T_TEST);
                } catch(InterruptedException ie) {}
            }
            gameFinished = false;
            gui.newGame();
            gui.setNewGameStarted(false);


            // an entire game
            while((!gui.isNewGameStarted())&&(!gameFinished)){
                if (!AI.canMakeMove()){
                    changeTurn();
                }

                if ((AI.getTurn() == GameField.WHITE)||(!AI.isSinglePlayer())) {
                    while (((move =  AI.getHumanMove()) == null) && (!gui.isNewGameStarted())) {
                        try {
                            Thread.sleep(T_TEST);
                        } catch(InterruptedException ie) {}
                    }
                    AI.setHumanMove(null);
                } else {
                    move = AI.suggest();
                    try {
                        Thread.sleep(T_PAUSE);
                    } catch (InterruptedException ie) {}
                    gui.setProgress(100);
                }
```

```
                if(gui.isNewGameStarted()){
                    break;
                }

                row = (byte)move.x;
                col = (byte)move.y;
                if (AI.isValidMove(row,col)) {
                    AI.update(row,col);
                    doMove(row,col);
                    changeTurn();
                } else if ((AI.getTurn() == GameField.BLACK)&&(AI.isSinglePlayer())) {
                    // AI cannot find a valid move
                    changeTurn();
                }

                if (AI.getTree().expandir(AI.getTurn()) == GameField.EMPTY) {
                        gui.finish(); // end of the game
                        gameFinished = true;
                }
            }
        }
    }

    private void doMove(int row, int col){
        final Point p = new Point(row,col);
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run(){
                    gui.doMove(p.x,p.y);
                }
            });
        } catch (Exception e){System.out.println(e);}
    }
    private void changeTurn() {
        AI.changeTurn();
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                gui.changeTurn(AI.getTurn());
            }
        });
    }

}
```

Código Fuente de la clase ficha, GameField:

```
public class GameField {
    /** The contents of a field */
    public static final byte EMPTY = 0;
    public static final byte WHITE = 1;
    public static final byte BLACK = 2;

    /** Creates a new instance of GameField */
    public GameField() {
    }
}
```

Código Fuente de la clase tablero, GameBoard:

```
public class GameBoard {
    private byte[][] v_board;

    /** Creates a new instance of GameBoard */
    public GameBoard() {
        GameBoardBuilder((byte)8,(byte)8);
    }

    public GameBoard(byte width) {
        GameBoardBuilder(width,width);
    }

    public GameBoard(byte width, byte height) {
        GameBoardBuilder(width,height);
    }

    private void GameBoardBuilder(byte width, byte height) {
        v_board = new byte[width][height];
    }

    public byte getWidth() {
        return (byte)v_board.length;
    }

    public byte getHeight() {
        if (v_board.length > 0)
```

```
            return (byte)v_board[0].length;
        else
            return 0;
    }

    public byte get(byte row, byte col) {
        return v_board[row][col];
    }

    public boolean isEmpty(byte row, byte col) {
        return v_board[row][col] == GameField.EMPTY;
    }

    public boolean isFirstPlayer(byte row, byte col) {
        return v_board[row][col] == GameField.WHITE;
    }

    public boolean isSecondPlayer(byte row, byte col) {
        return v_board[row][col] == GameField.BLACK;
    }

    public void empty(byte row, byte col) {
        v_board[row][col] = GameField.EMPTY;
    }

    public void setFirstPlayer(byte row, byte col) {
        v_board[row][col] = GameField.WHITE;
    }

    public void setSecondPlayer(byte row, byte col) {
        v_board[row][col] = GameField.BLACK;
    }

    /** Clone a board
     */
    public Object clone() {
        GameBoard b = new GameBoard(this.getWidth(), this.getHeight());

        for (byte row = 0; row < this.getHeight(); row++)
            for (byte col = 0; col < this.getWidth(); col++)
                  b.v_board[row][col] = this.v_board[row][col];
        return b;
    }

    /** Prints a board
     */
    public void print() {
        for (byte row = 0; row < this.getHeight(); row++){
            for (byte col = 0; col < this.getWidth(); col++)
                System.out.print(this.v_board[row][col]);
            System.out.println();
        }
    }

}
```

Código Fuente de la clase de la IA, ReversiAI:

```
import java.awt.*;
import java.util.*;

public class ReversiAI {

    private static byte[][] weights = {
        {120, -20, 20,  5,  5, 20, -20, 120},
        {-20, -40, -5, -5, -5, -5, -40, -20},
        { 20,  -5, 15,  3,  3, 15,  -5,  20},
        {  5,  -5,  3,  3,  3,  3,  -5,   5},
        {  5,  -5,  3,  3,  3,  3,  -5,   5},
        { 20,  -5, 15,  3,  3, 15,  -5,  20},
        {-20, -40, -5, -5, -5, -5, -40, -20},
        {120, -20, 20,  5,  5, 20, -20, 120}
    };

    private byte plys;
    private byte uplys;
    private ReversiTree tree;
    private byte turn;
    private Point humanMove;
    private boolean singlePlayer = true;
    private ReversiGUI gui;


    /** Creates a new instance of ReversiAI
     *  By default, a 8x8 board and the most simple
     *  depth for the algorithm is used.
     */
```

```
public ReversiAI() {
    ReversiAIbuilder((byte)8, (byte)8, (byte)-1);
}

/** Creates a new instance of ReversiAI with
 *  a heightxwidth board and p depth for the
 *  algorithm.
 */
public void ReversiAIbuilder(byte height, byte width, byte p) {
    plys = p;
    turn = GameField.WHITE;

    GameBoard data = new GameBoard(width,height);

    data.setFirstPlayer((byte)3,(byte)3);
    data.setFirstPlayer((byte)4,(byte)4);
    data.setSecondPlayer((byte)3,(byte)4);
    data.setSecondPlayer((byte)4,(byte)3);

    this.tree = new ReversiTree(null, data, null, width, height, turn, new Point(-1,-1));
}

/** Returns a table with the current game status
 */
public GameBoard getCurrentStatus() {
    return this.tree.getBoard();
}

/** Returns the current top-level node of the game tree
 */
public ReversiTree getTree() {
    return this.tree;
}

/** Returns board height
 */
public int getHeight() {
    return this.tree.getBoard().getHeight();
}

/** Returns board width
 */
public int getWidth() {
    return this.tree.getBoard().getWidth();
}

/** Returns the "plys" used in minimax algorithm by the AI
 */
public int getPlys() {
    return this.plys;
}

public void setHumanMove(Point p){
    this.humanMove = p;
}

public Point getHumanMove(){
    return this.humanMove;
}

/** Returns the current turn
 */
public byte getTurn() {
    return this.turn;
}

/** Changes player's turn to the next one
 */
public void changeTurn() {
    if (this.turn == GameField.WHITE)
        this.turn = GameField.BLACK;
    else
        this.turn = GameField.WHITE;
}

/** Set the "plys" to use by the AI
 */
public void setPlys(byte p) {
    this.plys  = p;
    this.uplys = p;
}

public boolean isSinglePlayer(){
    return this.singlePlayer;
}

public void setSinglePlayer(boolean p) {
    this.singlePlayer = p;
```

```
}

public void setGui(ReversiGUI g) {
    this.gui = g;
}

/** Determines wheter or not a move is valid within
 *  reversi game rules
 */
public boolean isValidMove(byte row, byte col) {
    Set s = new LinkedHashSet();
    return tree.isValidTableMove(row,col,this.turn,s);
}

/** returns whether or not a player could make at least one legal
 *  move
 */
public boolean canMakeMove() {
    ArrayList sons = this.tree.getSons();

    return (this.turn == this.tree.getTurn());
}

/** Update the game tree to reflect the current status
 *  when a move was done in (x,y).
 */
public void update(byte x, byte y) {
    ArrayList a = this.tree.getSons();
    ReversiTree n;
    Set s = new LinkedHashSet();
    boolean found = false;
    byte turn = this.tree.getTurn(), next;
    GameBoard b = (GameBoard)this.tree.getBoard().clone();

    if((a.isEmpty())||(!tree.getBoard().isEmpty(x,y)))
        return;

    /* Search the sons and take the one with the move
     * specified.
     */
    for(int i = 0; i < a.size(); i++){
        n = (ReversiTree)a.get(i);
        if (turn == n.getBoard().get(x,y)){
            tree = n;
            found = true;
            break;
        }
    }
    if (!found) {
        if (tree.isValidTableMove(x, y, turn, s)){
            if (turn == GameField.WHITE) {
                next = GameField.BLACK;
                b.setFirstPlayer(x,y);
            } else {
                next = GameField.WHITE;
                b.setSecondPlayer(x,y);
            }
            Iterator it = s.iterator();
            while (it.hasNext()) {
                Point p = (Point)it.next();
                x = (byte)p.getX();
                y = (byte)p.getY();
                if (b.isFirstPlayer(x,y))
                    b.setSecondPlayer(x,y);
                else
                    b.setFirstPlayer(x,y);
            }
            this.tree = new ReversiTree(this.tree,b,s,
                                        b.getWidth(),b.getHeight(),
                                        next,new Point(x,y));
            this.tree.expandir(next);
        }
    }

    System.gc();    // invoke garbage collector
    return;
}

/** Suggest the best possible move given the current game status
 */
public Point suggest() {
    int sc, g = 0;
    ArrayList l;
    Iterator i;
    ReversiTree n;

    if (uplys > plys)
        plys++; // if the ply has been reduced because of memory
```

```
                // usage, try to restore it slowly
        sc = minimax(this.tree, 0, (short)-(Short.MAX_VALUE),Short.MAX_VALUE);
        l = this.tree.getSons();
        if (l.isEmpty()) {
            return new Point(-1,-1);
        }

        i = l.iterator();
        while (i.hasNext()) {
            n = (ReversiTree)(i.next());

            if (n.getScore() == sc)
                return n.getMove();
            g++;
        }
        return new Point(-1,-1);
    }


    private short max(short a, short b){
        if (a > b)
            return a;
        return b;
    }

    private short min(short a, short b){
        if (a < b)
            return a;
        return b;
    }
    private short minimax(ReversiTree tree, int depth,short alpha, short beta) {

        Iterator i;
        ReversiTree n;
        short s = 0, ex = 0;

        try {
            if (plys > depth) {
                // tree has to grow
                if (tree.expandir(tree.getTurn()) != GameField.EMPTY) {
                    i = tree.getSons().iterator();
                    while (i.hasNext()) {
                        // we assume the AI plays with BLACK
                        if (tree.getTurn() == GameField.BLACK){ // MAX
                            n = (ReversiTree)(i.next());
                            s = minimax(n, depth + 1,alpha, beta);
                            if ((s > ex)||(ex == 0))
                                ex = s;
                            alpha = max(alpha, ex);
                            if (alpha >= beta){
                                ArrayList sons = new ArrayList();
                                sons.add(n);
                                tree.setSons(sons);
                                tree.setScore(ex);
                                return ex;
                            }
                        } else { // MIN
                            n = (ReversiTree)(i.next());
                            s = minimax(n, depth + 1,alpha, beta);
                            if ((s <= ex)||(ex == 0))
                                ex = s;
                            beta = min(ex,beta);
                            if (beta <= alpha){
                                ArrayList sons = new ArrayList();
                                sons.add(n);
                                tree.setScore(ex);
                                tree.setSons(sons);
                                return ex;
                            }
                        }
                    }
                } else {
                    ex = e(tree);
                    if (!tree.isFinal())
                        tree.getFather().getSons().remove(this);
                }
            } else {
                ex = e(tree);
                if (!tree.isFinal())
                    tree.getFather().getSons().remove(this);
            }
        } catch (java.lang.OutOfMemoryError e) {
            this.plys--;
            ArrayList l = new ArrayList();
            tree.setSons(l);
            System.gc();
            if (ex == 0)
                ex = e(tree);
```

```
        }
        tree.setScore(ex);

        if (gui.getProgress() < 100/(depth+1))
            gui.setProgress(100/(depth+1));

        return ex;
    }

    /* Evaluation function for a board and a player.
     * Evaluation consists in adding player's token weights
     * and subtracting opponent one's.
     */
    private short e(ReversiTree node){
        byte mine, other, mine_border, other_border, mine_corner, other_corner;
        int m_weight, o_weight;

        m_weight = o_weight = mine = other = mine_border = other_border = mine_corner = other_corner
= 0;
        for (byte i = 0; i < getHeight(); i++) {
            for(byte j = 0; j < getWidth(); j++) {
                // borders
                if (node.getBoard().get(i,j) == GameField.BLACK &&
                    (i == 1 || i == getHeight() - 1 || j == 1 || j == getWidth() - 1))
                    mine_border++;
                if (node.getBoard().get(i,j) == GameField.WHITE &&
                    (i == 1 || i == getHeight() - 1 || j == 1 || j == getWidth() - 1))
                    other_border++;
                // corners
                if (node.getBoard().get(i,j) == GameField.BLACK &&
                    ((i == 1 && j == 1) || (i == 1 && j == getWidth() - 1) ||
                     (i == getHeight() - 1 && j == 1) ||
                     (i == getHeight() - 1 && j == getWidth() - 1)))
                    mine_corner++;
                if (node.getBoard().get(i,j) == GameField.WHITE &&
                    ((i == 1 && j == 1) || (i == 1 && j == getWidth() - 1) ||
                     (i == getHeight() - 1 && j == 1) ||
                     (i == getHeight() - 1 && j == getWidth() - 1)))
                    other_corner++;
                // totals
                if (node.getBoard().get(i,j) == GameField.BLACK) {
                    m_weight += weights[i][j];
                    mine++;
                }
                if (node.getBoard().get(i,j) == GameField.WHITE) {
                    o_weight += weights[i][j];
                    other++;
                }
            }
        }
        if (node.isFinal() && mine > other) {
            return Short.MAX_VALUE;
        }
        if (node.isFinal() && (mine < other)) {
            return Short.MIN_VALUE;
        }

        return (short)(m_weight + 1.5*m_weight*mine_border + 3*m_weight*mine_corner -
               -o_weight - 1.5*o_weight*other_border - 3*o_weight*other_corner);
    }
}
```

Código Fuente de la clase de control de la GUI, ReversiGUI:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.text.*;

public class ReversiGUI extends javax.swing.JFrame {

    /** The table model is used to represent data in the table.
     *  This class extends the functionality of the default one by
     *  substituting the objects in the cells by images which can be
     *  changed in runtime.
     */
    class ReversiTableModel extends javax.swing.table.AbstractTableModel {
        // basic states
        private ImageIcon empty = new ImageIcon("img/empty.gif","empty");
        private ImageIcon black = new ImageIcon("img/black.gif","black");
        private ImageIcon white = new ImageIcon("img/white.gif","white");
        // for common animations
        private ImageIcon unscaled_w2b = new ImageIcon("img/w2b.gif","black");
        private ImageIcon unscaled_b2w = new ImageIcon("img/b2w.gif", "white");
```

```
private ImageIcon w2b;
private ImageIcon b2w;

private String[] columnNames = {"","","","","","","","",""};
private Object[][] data = {
    {empty,empty,empty,empty,empty,empty,empty,empty},
    {empty,empty,empty,empty,empty,empty,empty,empty},
    {empty,empty,empty,empty,empty,empty,empty,empty},
    {empty,empty,empty,white,black,empty,empty,empty},
    {empty,empty,empty,black,white,empty,empty,empty},
    {empty,empty,empty,empty,empty,empty,empty,empty},
    {empty,empty,empty,empty,empty,empty,empty,empty},
    {empty,empty,empty,empty,empty,empty,empty,empty}
};

public int getColumnCount() {
    return data.length;
}

public int getRowCount() {
    return data.length;
}

public String getColumnName(int col) {
    return columnNames[col];
}

public Object getValueAt(int row, int col) {
    return data[row][col];
}

public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

public boolean isCellEditable(int row, int col) {
    return false;
}

public void setValueAt(Object value, int row, int col) {
    data[row][col] = value;
    fireTableCellUpdated(row,col);
}

public void setWhite(int row, int col) {
    int width = getWidth() / data.length /2;
    if (((ImageIcon)data[row][col]).getDescription() == "empty") {
        setValueAt(white,row,col);
    } else if (((ImageIcon)data[row][col]).getDescription() == "black") {
        b2w = new ImageIcon(unscaled_b2w.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        b2w.setImageObserver(iobs);
        b2w.setDescription("white");
        setValueAt(b2w,row,col);
    }
}

public void setBlack(int row, int col) {
    int width = getWidth() / data.length /2;
    if (((ImageIcon)data[row][col]).getDescription() == "empty") {
        setValueAt(black,row,col);
    } else if (((ImageIcon)data[row][col]).getDescription() == "white") {
        w2b = new ImageIcon(unscaled_w2b.getImage().getScaledInstance(width, width,
                Image.SCALE_FAST));
        w2b.setImageObserver(iobs);
        w2b.setDescription("black");
        setValueAt(w2b,row,col);
    }
}

public void scaleImages() {
    int width = getWidth() / data.length /2;
    empty.setImage(empty.getImage().getScaledInstance(width, width,
            Image.SCALE_FAST));
    black.setImage(black.getImage().getScaledInstance(width, width,
            Image.SCALE_FAST));
    white.setImage(white.getImage().getScaledInstance(width, width,
            Image.SCALE_FAST));
    b2w = new ImageIcon(unscaled_b2w.getImage().getScaledInstance(width, width,
            Image.SCALE_FAST));
    b2w.setImageObserver(iobs);
    w2b = new ImageIcon(unscaled_w2b.getImage().getScaledInstance(width, width,
            Image.SCALE_FAST));
    w2b.setImageObserver(iobs);
}

public boolean isEmpty(int row, int col) {
    return ((ImageIcon)data[row][col]).getDescription() == "empty";
```

```
        }

        public boolean isBlack(int row, int col) {
            return ((ImageIcon)data[row][col]).getDescription() == "black";
        }

        public boolean isWhite(int row, int col) {
            return ((ImageIcon)data[row][col]).getDescription() == "white";
        }

        public void reset() {
            for (int i = 0; i < getRowCount(); i++)
                for (int j = 0; j < getColumnCount(); j++)
                    data[i][j] = empty;
            data[3][3] = white;
            data[3][4] = black;
            data[4][3] = black;
            data[4][4] = white;

        }

        public ImageObserver iobs = new ImageObserver() {
            public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
                repaint();
                return true;
            }};
    };

    /** Creates new form ReversiGUI */
    public ReversiGUI(ReversiAI rai) {
        ReversiTableModel model = new ReversiTableModel();
        AI = rai;

        initComponents();
        this.setLocationRelativeTo(null);
        setVisible(true);
        timeListener = new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                /***************************** This is a dirty hack!
                 * Date and Time methods seem to need negative values
                 * to represent the first hour, so an effective value
                 * of 1000 miliseconds translates into an hour and 1
                 * second, wherever a single second was expected.
                 */
                java.sql.Time time = new java.sql.Time(t -1000*60*60);
                t += 1000;
                TimeLabel.setText(time.toString());
            }
        };
        GameTimer = new javax.swing.Timer(1000,timeListener);
        PlayerDialog.setSize(200,160);

        model.scaleImages();
        Board.setModel(model);
        Board.setRowHeight(Board.getWidth() / Board.getColumnModel().getColumnCount());
        initBoard();
    }

    /** This method is called at the begining and when user requests
     *  a new game to reset and initialize the board.
     */
    private void initBoard() {
        ReversiTableModel model;

        model = (ReversiTableModel)Board.getModel();

        /* Initialize board */
        model.reset();
        Board.setEnabled(false);
        ComputerScore.setText("0");
        PlayerScore.setText("0");
    }

    public void newGame(){
        byte plys = 1;
        boolean singlePlayer = true;

        initBoard();

        // game mode selection
        Object[] numplayers = {"Single player", "Two players"};
        String s = (String)JOptionPane.showInputDialog(null,
                "Select the game mode you want to play:",
                "Game mode",
                JOptionPane.PLAIN_MESSAGE,
                null,numplayers,"Single player");

        if (s == "Two players") {
```

```
            singlePlayer = false;
            jLabel1.setText("Player 2");
            jComboBox1.setEnabled(true);
            if (jLabel2.getText() == "Player")
                jLabel2.setText("Player 1");
        } else if (s == "Single player") {
            singlePlayer = true;
            jLabel1.setText("Computer");
            jComboBox1.setEnabled(false);
        } else return;

        if (singlePlayer) {
            // level selection
            Object[] levels = {"easy", "medium", "hard"};
            s =(String)JOptionPane.showInputDialog(null,
                    "Select your preferred dificulty level:",
                    "Dificulty level",
                    JOptionPane.PLAIN_MESSAGE,
                    null,levels,"easy");
            if (s == "hard")
                plys = 7;
            else if (s == "medium")
                plys = 4;
            else if (s == "easy")
                plys = 1;
            else return;
            jLabel3.setText(s+" level");
        }

        AI.ReversiAIbuilder((byte)8,(byte)8,plys);
        AI.setSinglePlayer(singlePlayer);
        // 1st legal moves
        AI.getTree().expandir(AI.getTurn());

        Board.setEnabled(true);

        ComputerScore.setText("2");
        PlayerScore.setText("2");

        //timer
        t = 1000;
        GameTimer.restart();

        // paint turn colours
        changeTurn(GameField.WHITE);
    }

    public void setNewGameStarted(boolean v){
        newgamestarted = v;
    }

    public boolean isNewGameStarted(){
        return newgamestarted;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        PlayerDialog = new javax.swing.JDialog();
        AcceptNameButton = new javax.swing.JButton();
        jLabel23 = new javax.swing.JLabel();
        playerName = new javax.swing.JTextField();
        jComboBox1 = new javax.swing.JComboBox();
        jPanel1 = new javax.swing.JPanel();
        Board = new javax.swing.JTable();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        jLabel12 = new javax.swing.JLabel();
        jLabel13 = new javax.swing.JLabel();
        jLabel14 = new javax.swing.JLabel();
        jLabel15 = new javax.swing.JLabel();
        jLabel16 = new javax.swing.JLabel();
        jLabel17 = new javax.swing.JLabel();
        jLabel18 = new javax.swing.JLabel();
        jLabel19 = new javax.swing.JLabel();
        jLabel20 = new javax.swing.JLabel();
        jLabel21 = new javax.swing.JLabel();
```

```
        jLabel22 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jPanel3 = new javax.swing.JPanel();
        ComputerScore = new javax.swing.JLabel();
        jPanel4 = new javax.swing.JPanel();
        PlayerScore = new javax.swing.JLabel();
        TimeLabel = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jSeparator1 = new javax.swing.JSeparator();
        jProgressBar1 = new javax.swing.JProgressBar();
        jMenuBar1 = new javax.swing.JMenuBar();
        GameMenu = new javax.swing.JMenu();
        MenuNew = new javax.swing.JMenuItem();
        MenuPlayer = new javax.swing.JMenuItem();
        jSeparator2 = new javax.swing.JSeparator();
        MenuExit = new javax.swing.JMenuItem();
        HelpMenu = new javax.swing.JMenu();
        MenuAbout = new javax.swing.JMenuItem();

        PlayerDialog.getContentPane().setLayout(new java.awt.GridBagLayout());

        PlayerDialog.setTitle("Change name");
        PlayerDialog.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        PlayerDialog.setResizable(false);
        AcceptNameButton.setText("Accept");
        AcceptNameButton.setMinimumSize(new java.awt.Dimension(50, 10));
        AcceptNameButton.setPreferredSize(new java.awt.Dimension(50, 10));
        AcceptNameButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AcceptNameButtonActionPerformed(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 50;
        gridBagConstraints.ipady = 15;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.insets = new java.awt.Insets(0, 30, 20, 30);
        PlayerDialog.getContentPane().add(AcceptNameButton, gridBagConstraints);

        jLabel23.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel23.setText("Please, enter your name:");
        jLabel23.setMinimumSize(new java.awt.Dimension(100, 10));
        jLabel23.setPreferredSize(new java.awt.Dimension(100, 10));
        jLabel23.setRequestFocusEnabled(false);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 70;
        gridBagConstraints.ipady = 10;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.insets = new java.awt.Insets(20, 10, 10, 10);
        PlayerDialog.getContentPane().add(jLabel23, gridBagConstraints);

        playerName.setText(jLabel2.getText());
        playerName.setMinimumSize(new java.awt.Dimension(100, 10));
        playerName.setPreferredSize(new java.awt.Dimension(100, 10));
        playerName.setRequestFocusEnabled(false);
        playerName.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                playerNameActionPerformed(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 70;
        gridBagConstraints.ipady = 10;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.insets = new java.awt.Insets(0, 10, 10, 10);
        PlayerDialog.getContentPane().add(playerName, gridBagConstraints);

         jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Player 1", "Player
2" }));
        jComboBox1.setEnabled(false);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 50;
```

```
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.insets = new java.awt.Insets(0, 10, 10, 10);
        PlayerDialog.getContentPane().add(jComboBox1, gridBagConstraints);

        getContentPane().setLayout(new java.awt.GridBagLayout());

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("jReversi");
        setName("MainFrame");
        setResizable(false);
        jPanel1.setLayout(null);

        jPanel1.setBorder(new javax.swing.border.TitledBorder("Board"));
        jPanel1.setName("BoardPanel");
        Board.setBackground(new java.awt.Color(0, 92, 0));
        Board.setBorder(new javax.swing.border.BevelBorder(javax.swing.border.BevelBorder.LOWERED));
        Board.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null},
                {null, null, null, null, null, null, null, null}
            },
            new String [] {
                 "Title 1", "Title 2", "Title 3", "Title 4", "Title 5", "Title 6", "Title 7", "Title
8"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, false, false, false, false, false, false, false
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Board.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        Board.setAutoscrolls(false);
        Board.setDoubleBuffered(true);
        Board.setEnabled(false);
        Board.setGridColor(new java.awt.Color(0, 51, 0));
        Board.setIntercellSpacing(new java.awt.Dimension(0, 0));
        Board.setMaximumSize(new java.awt.Dimension(400, 400));
        Board.setMinimumSize(new java.awt.Dimension(400, 400));
        Board.setName("Panel");
        Board.setPreferredSize(new java.awt.Dimension(250, 250));
        Board.setRowSelectionAllowed(false);
        Board.setSelectionBackground(new java.awt.Color(255, 255, 255));
        Board.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                BoardMouseClicked(evt);
            }
        });

        jPanel1.add(Board);
        Board.setBounds(20, 30, 250, 250);

        jLabel5.setFont(new java.awt.Font("Courier", 1, 10));
        jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel5.setText("A    B    C    D    E    F    G    H");
        jLabel5.setFocusable(false);
        jLabel5.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        jPanel1.add(jLabel5);
        jLabel5.setBounds(20, 10, 250, 20);

        jLabel6.setFont(new java.awt.Font("Courier", 1, 10));
        jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel6.setText("A    B    C    D    E    F    G    H");
        jLabel6.setFocusable(false);
        jLabel6.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        jPanel1.add(jLabel6);
        jLabel6.setBounds(20, 280, 250, 20);

        jLabel7.setFont(new java.awt.Font("Courier", 1, 10));
        jLabel7.setText("1");
        jPanel1.add(jLabel7);
        jLabel7.setBounds(275, 35, 10, 20);

        jLabel8.setFont(new java.awt.Font("Courier", 1, 10));
        jLabel8.setText("2");
        jPanel1.add(jLabel8);
        jLabel8.setBounds(275, 66, 10, 20);

        jLabel9.setFont(new java.awt.Font("Courier", 1, 10));
```

```
jLabel9.setText("3");
jPanel1.add(jLabel9);
jLabel9.setBounds(275, 97, 10, 20);

jLabel10.setFont(new java.awt.Font("Courier", 1, 10));
jLabel10.setText("4");
jPanel1.add(jLabel10);
jLabel10.setBounds(275, 128, 10, 20);

jLabel11.setFont(new java.awt.Font("Courier", 1, 10));
jLabel11.setText("5");
jPanel1.add(jLabel11);
jLabel11.setBounds(275, 159, 10, 20);

jLabel12.setFont(new java.awt.Font("Courier", 1, 10));
jLabel12.setText("6");
jPanel1.add(jLabel12);
jLabel12.setBounds(275, 190, 10, 20);

jLabel13.setFont(new java.awt.Font("Courier", 1, 10));
jLabel13.setText("7");
jPanel1.add(jLabel13);
jLabel13.setBounds(275, 221, 10, 20);

jLabel14.setFont(new java.awt.Font("Courier", 1, 10));
jLabel14.setText("8");
jPanel1.add(jLabel14);
jLabel14.setBounds(275, 252, 10, 20);

jLabel15.setFont(new java.awt.Font("Courier", 1, 10));
jLabel15.setText("8");
jPanel1.add(jLabel15);
jLabel15.setBounds(9, 252, 10, 20);

jLabel16.setFont(new java.awt.Font("Courier", 1, 10));
jLabel16.setText("7");
jPanel1.add(jLabel16);
jLabel16.setBounds(9, 221, 10, 20);

jLabel17.setFont(new java.awt.Font("Courier", 1, 10));
jLabel17.setText("6");
jPanel1.add(jLabel17);
jLabel17.setBounds(9, 190, 10, 20);

jLabel18.setFont(new java.awt.Font("Courier", 1, 10));
jLabel18.setText("5");
jPanel1.add(jLabel18);
jLabel18.setBounds(9, 159, 10, 20);

jLabel19.setFont(new java.awt.Font("Courier", 1, 10));
jLabel19.setText("4");
jPanel1.add(jLabel19);
jLabel19.setBounds(9, 128, 10, 20);

jLabel20.setFont(new java.awt.Font("Courier", 1, 10));
jLabel20.setText("3");
jPanel1.add(jLabel20);
jLabel20.setBounds(9, 97, 10, 20);

jLabel21.setFont(new java.awt.Font("Courier", 1, 10));
jLabel21.setText("2");
jPanel1.add(jLabel21);
jLabel21.setBounds(9, 66, 10, 20);

jLabel22.setFont(new java.awt.Font("Courier", 1, 10));
jLabel22.setText("1");
jPanel1.add(jLabel22);
jLabel22.setBounds(9, 35, 10, 20);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipadx = 289;
gridBagConstraints.ipady = 299;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(jPanel1, gridBagConstraints);

jPanel2.setLayout(null);

jPanel2.setBorder(new javax.swing.border.TitledBorder("Score"));
jPanel2.setName("ScorePanel");
jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel1.setText("Computer");
jPanel2.add(jLabel1);
jLabel1.setBounds(10, 20, 130, 20);

jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
```

```
        jLabel2.setText("Player");
        jPanel2.add(jLabel2);
        jLabel2.setBounds(10, 90, 130, 20);

        jPanel3.setLayout(null);

        jPanel3.setBackground(new java.awt.Color(0, 0, 0));
        jPanel3.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(0, 0, 0), 1, true));
        ComputerScore.setBackground(new java.awt.Color(0, 0, 0));
        ComputerScore.setFont(new java.awt.Font("Courier", 0, 14));
        ComputerScore.setForeground(new java.awt.Color(255, 255, 255));
        ComputerScore.setText("0");
        jPanel3.add(ComputerScore);
        ComputerScore.setBounds(4, 6, 30, 20);

        jPanel2.add(jPanel3);
        jPanel3.setBounds(55, 50, 40, 30);

        jPanel4.setLayout(null);

        jPanel4.setBackground(new java.awt.Color(255, 255, 255));
        jPanel4.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(0, 0, 0), 1, true));
        PlayerScore.setFont(new java.awt.Font("Courier", 0, 14));
        PlayerScore.setText("0");
        jPanel4.add(PlayerScore);
        PlayerScore.setBounds(4, 6, 30, 20);

        jPanel2.add(jPanel4);
        jPanel4.setBounds(55, 120, 40, 30);

        TimeLabel.setFont(new java.awt.Font("Courier", 1, 14));
        TimeLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        TimeLabel.setText("00:00:00");
        jPanel2.add(TimeLabel);
        TimeLabel.setBounds(5, 274, 140, 20);

        jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jPanel2.add(jLabel3);
        jLabel3.setBounds(10, 220, 130, 20);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipady = 100;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        getContentPane().add(jPanel2, gridBagConstraints);

                                        jSeparator1.setBorder(new     javax.swing.border.BevelBorder
(javax.swing.border.BevelBorder.RAISED));
        jSeparator1.setName("Separator");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 289;
        gridBagConstraints.ipady = 2;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        getContentPane().add(jSeparator1, gridBagConstraints);

        jProgressBar1.setName("ProgressBar");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.gridwidth = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 289;
        gridBagConstraints.ipady = 6;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.insets = new java.awt.Insets(4, 0, 2, 0);
        getContentPane().add(jProgressBar1, gridBagConstraints);

        GameMenu.setMnemonic('G');
        GameMenu.setText("Game");
        GameMenu.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                GameMenuActionPerformed(evt);
            }
        });

        MenuNew.setMnemonic('N');
        MenuNew.setText("New Game");
        MenuNew.setName("MenuNew");
        MenuNew.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MenuNewActionPerformed(evt);
            }
```

```
        });

        GameMenu.add(MenuNew);

        MenuPlayer.setMnemonic('P');
        MenuPlayer.setText("Player name");
        MenuPlayer.setName("MenuPlayer");
        MenuPlayer.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MenuPlayerActionPerformed(evt);
            }
        });

        GameMenu.add(MenuPlayer);

        GameMenu.add(jSeparator2);

        MenuExit.setMnemonic('Q');
        MenuExit.setText("Quit");
        MenuExit.setName("MenuExit");
        MenuExit.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MenuExitActionPerformed(evt);
            }
        });

        GameMenu.add(MenuExit);

        jMenuBar1.add(GameMenu);

        HelpMenu.setMnemonic('H');
        HelpMenu.setText("Help");

        MenuAbout.setMnemonic('A');
        MenuAbout.setText("About");
        MenuAbout.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                HelpAbout(evt);
            }
        });

        HelpMenu.add(MenuAbout);

        jMenuBar1.add(HelpMenu);

        setJMenuBar(jMenuBar1);

        pack();
    }
    // </editor-fold>

    private void GameMenuActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    }

    private void HelpAbout(java.awt.event.ActionEvent evt) {
        JOptionPane.showMessageDialog(null,
                "jReversi developers are:\n" +
                "* Tomás Aguado Gómez\n" +
                "* Angel Jara Gómez\n" +
                "* Jaime Pérez Crespo",
                "About jReversi", JOptionPane.INFORMATION_MESSAGE);
    }


    private void MenuNewActionPerformed(java.awt.event.ActionEvent evt) {
        newgamestarted = true;
    }

    private void playerNameActionPerformed(java.awt.event.ActionEvent evt) {
        if (playerName.getText().length() != 0) {
            if (jComboBox1.getSelectedItem() == "Player 1")
                jLabel2.setText(playerName.getText());
            else
                jLabel1.setText(playerName.getText());

            PlayerDialog.hide();
        }
    }

    private void AcceptNameButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if (playerName.getText().length() != 0) {
            if (jComboBox1.getSelectedItem() == "Player 1")
                jLabel2.setText(playerName.getText());
            else
                jLabel1.setText(playerName.getText());

            PlayerDialog.hide();
```

```
        }
    }

    private void MenuPlayerActionPerformed(java.awt.event.ActionEvent evt) {
        playerName.setText(jLabel2.getText());
        PlayerDialog.setLocationRelativeTo(this);
        PlayerDialog.show();
    }

    private void MenuExitActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }

    public void doMove(int row, int col) {
        ReversiTableModel m = (ReversiTableModel)Board.getModel();
        int score;
        int t = AI.getTurn();

        if (t == GameField.WHITE) {
            score = Integer.parseInt(PlayerScore.getText());
            PlayerScore.setText(Integer.toString(++score));
            m.setWhite(row,col);
        } else {
            score = Integer.parseInt(ComputerScore.getText());
            ComputerScore.setText(Integer.toString(++score));
            m.setBlack(row,col);
        }
        Iterator i = AI.getTree().getTakes().iterator();
        while (i.hasNext()) {
            Point p = (Point)i.next();
            if (m.isWhite((int)p.getX(),(int)p.getY())) {
                score = Integer.parseInt(ComputerScore.getText());
                ComputerScore.setText(Integer.toString(++score));
                score = Integer.parseInt(PlayerScore.getText());
                PlayerScore.setText(Integer.toString(--score));
                m.setBlack((int)p.getX(),(int)p.getY());
            } else {
                score = Integer.parseInt(PlayerScore.getText());
                PlayerScore.setText(Integer.toString(++score));
                score = Integer.parseInt(ComputerScore.getText());
                ComputerScore.setText(Integer.toString(--score));
                m.setWhite((int)p.getX(),(int)p.getY());
            }
        }
    }

    private void setToolTip(int col, int row) {
        char c = 'A';

        c = (char)((int) c + col);
        Board.setToolTipText("Last move: "
                +Character.toString(c)
                +"/"+String.valueOf(row+1));
    }

    public void setProgress(int n) {
        jProgressBar1.setValue(n);
    }

    public int getProgress() {
        return jProgressBar1.getValue();
    }

    public void changeTurn(int turn) {
        // change the current turn in the GUI
        if (turn == GameField.WHITE) {
            jLabel2.setForeground(Color.RED);
            jLabel1.setForeground(Color.BLACK);
        } else {
            jLabel1.setForeground(Color.RED);
            jLabel2.setForeground(Color.BLACK);
        }
    }

    public void finish() {
        int black = Integer.parseInt(ComputerScore.getText());
        int white = Integer.parseInt(PlayerScore.getText());

        GameTimer.stop();
        jLabel1.setForeground(Color.BLACK);
        jLabel2.setForeground(Color.BLACK);
        Board.setEnabled(false);
        if (black > white) {
            JOptionPane.showMessageDialog(null,"Game over, "+
                    jLabel1.getText()+" wins!","Game over", JOptionPane.YES_OPTION);
            return;
        }
        if (white > black) {
```

```
            JOptionPane.showMessageDialog(null,"Game over, "+
                    jLabel2.getText()+" wins!","Game over", JOptionPane.YES_OPTION);
            return;
        }
        JOptionPane.showMessageDialog(null,"Game over, "+
                "no one wins!","Game over", JOptionPane.YES_OPTION);
    }

    private void BoardMouseClicked(java.awt.event.MouseEvent evt) {
        int col = Board.getSelectedColumn();
        int row = Board.getSelectedRow();
        Point p;
        int t;

        // board active only if in human player's turn
        if ((Board.isEnabled())&&((AI.getTurn() == GameField.WHITE)||(!AI.isSinglePlayer()))) {
            AI.setHumanMove(new Point(row,col));
            jProgressBar1.setValue(0);
        }
    }

    // Variables declaration - do not modify
    private javax.swing.JButton AcceptNameButton;
    private javax.swing.JTable Board;
    private javax.swing.JLabel ComputerScore;
    private javax.swing.JMenu GameMenu;
    private javax.swing.JMenu HelpMenu;
    private javax.swing.JMenuItem MenuAbout;
    private javax.swing.JMenuItem MenuExit;
    private javax.swing.JMenuItem MenuHelp;
    private javax.swing.JMenuItem MenuNew;
    private javax.swing.JMenuItem MenuPlayer;
    private javax.swing.JDialog PlayerDialog;
    private javax.swing.JLabel PlayerScore;
    private javax.swing.JLabel TimeLabel;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel23;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JProgressBar jProgressBar1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JTextField playerName;
    private javax.swing.Timer GameTimer;
    private long t;
    private ActionListener timeListener;
    private boolean singlePlayer = true;
    private boolean newgamestarted = false;
    private ReversiAI AI;
    // End of variables declaration
}
```