

# Práctica 5

## Guía de Diseño

# IMPORTANTE

El diseño proporcionado en estas transparencias es sólo una manera de lograr nuestro objetivo. No es necesariamente el mejor en general, pero es lo suficientemente bueno teniendo en cuenta los requisitos de la práctica 5 .

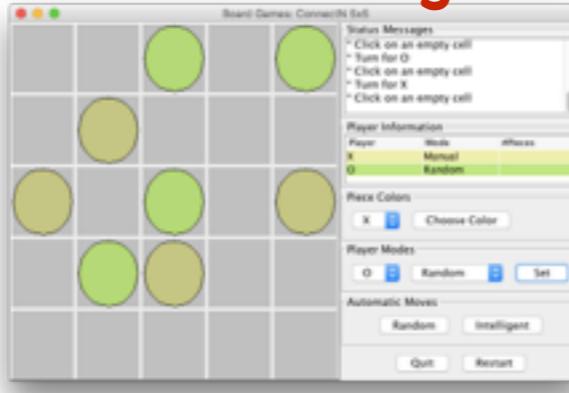
No es obligatorio seguir exactamente este diseño, pero tienes que justificar (durante la defensa de la práctica) cualquier diseño que decides usar.

# Vistas para los Juegos de Tablero

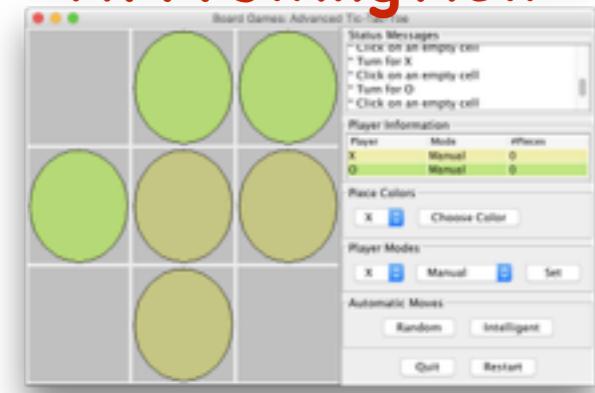
AtaxxSwipeView



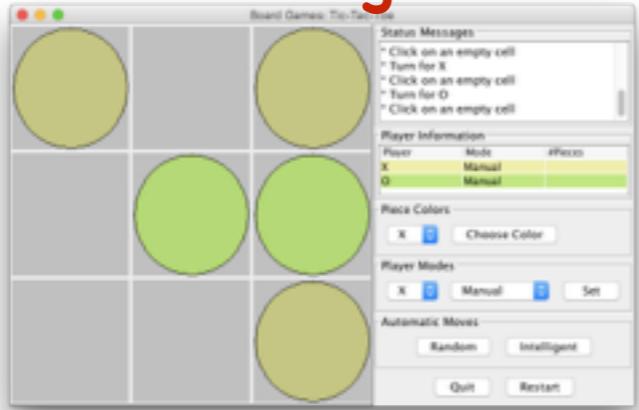
ConnectNSwipeView



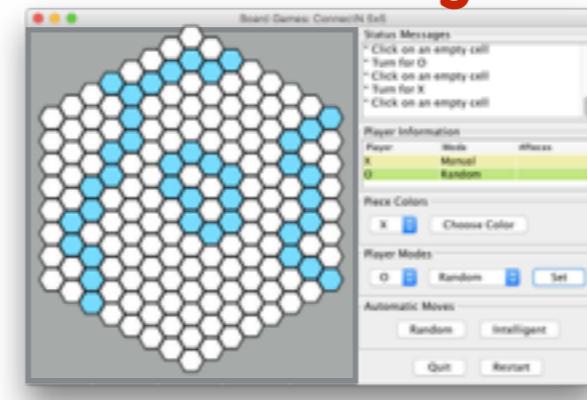
ATTTSwipeView



TTTSwipeView



HavannahSwipeView



es sólo un ejemplo, no  
hay que implementar  
en la pr5.

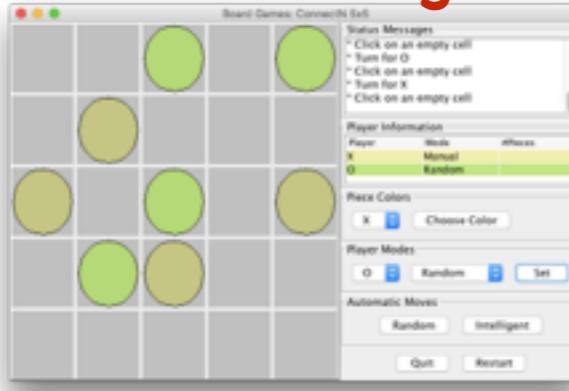
Tenemos que desarrollar vistas para varios juegos, sin embargo, estas vistas tienen mucho en común. Nuestro objetivo es usar un diseño que tiene en cuenta estas partes comunes, por lo facilita el desarrollo de vista para otros juegos, etc.

# Identificar Relaciones: I

AtaxxSwingView



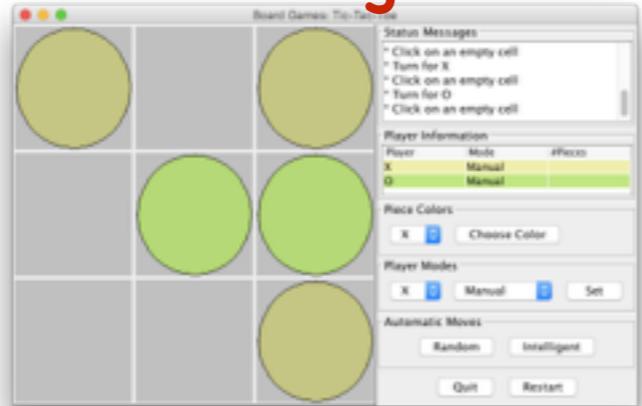
ConnectNSwingView



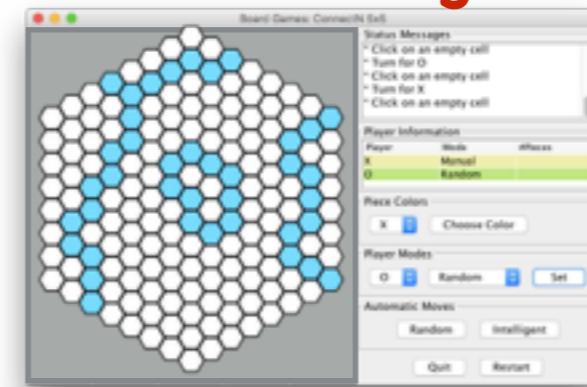
ATTT SwingView



TTTSwingView



Havannah SwingView

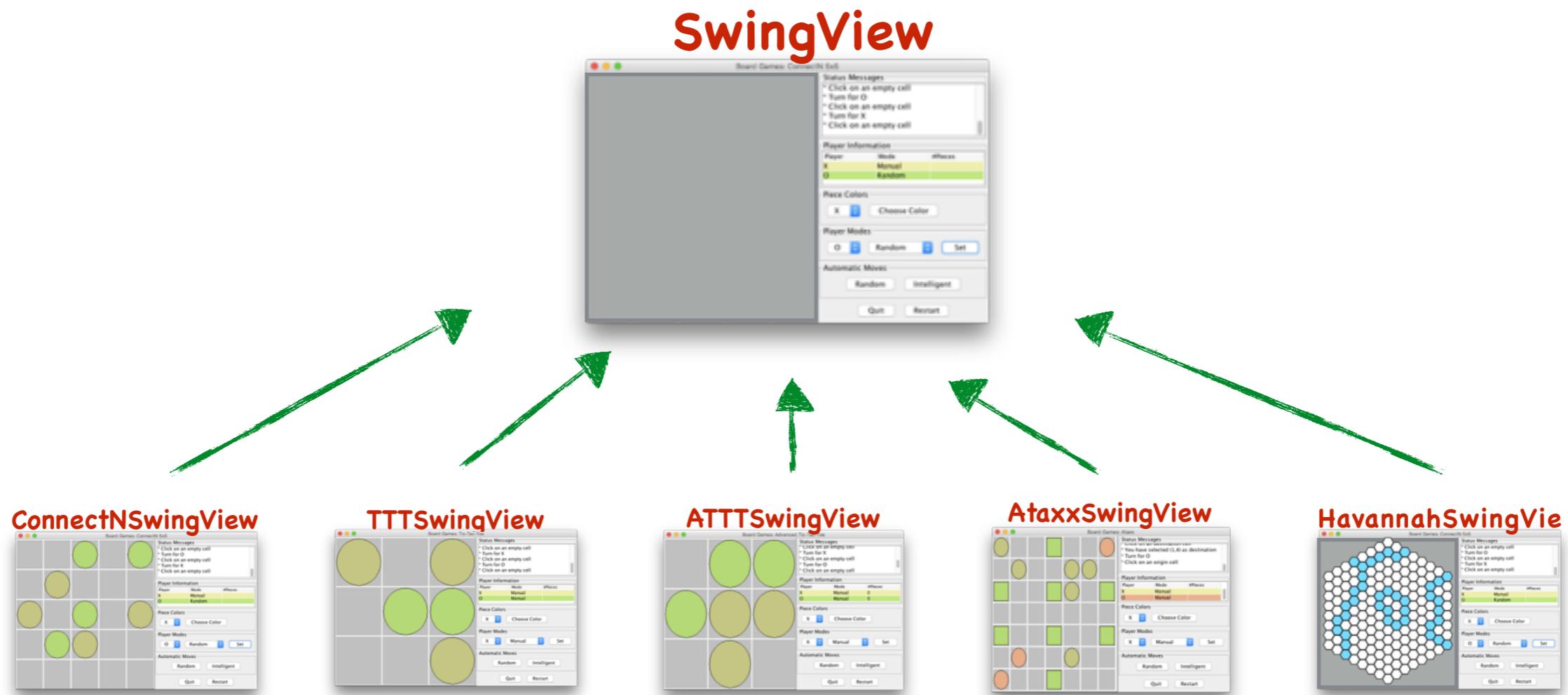


es sólo un ejemplo, no  
hay que implementar  
en la pr5.

- ◆ La parte visual de la vista se puede dividir en: control y tablero.
- ◆ La parte del control incluye una parte gráfica y una funcionalidad correspondiente (jugar automáticamente , cambiar colores , etc).
- ◆ La parte del tablero incluye una parte gráfica y una funcionalidad correspondiente (la forma en que introducimos movimientos).
- ◆ La parte del control es común para todos los juegos, el tablero puede ser diferente de un juego a otro ...

# Identificar Relaciones: II

Podemos usar un diseño en el que se define la parte gráfica del control y su funcionalidad en una clase SwingView ...



... y cada vista de juego hereda la funcionalidad de **SwingView** y define la parte gráfica del tablero y su funcionalidad correspondiente ...

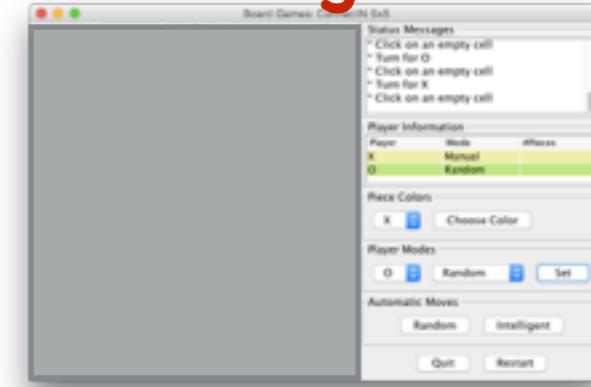
# Identificar Relaciones: III

Todos los juegos con tablero rectangular comparten la parte gráfica del tablero, pero pueden tener formas diferentes para hacer movimientos ya que los movimientos son diferentes ...

Para que el diseño tenga esta relación en cuenta introducimos otro nivel en la jerarquía de clases ...

La vista TTT Swing View es la misma que ConnectN Swing View, así que no hay que implementar TTT Swing View, sino usar ConnectN Swing View en la factoría de TTT ...

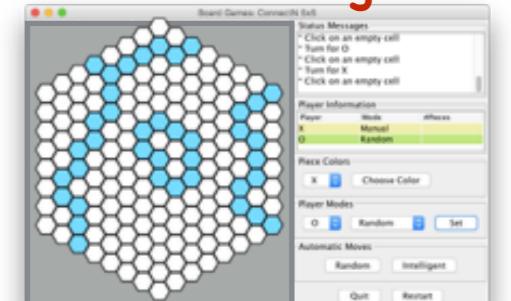
SwingView



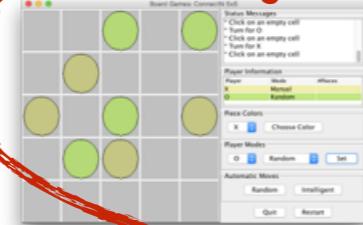
RectBoard Swing View



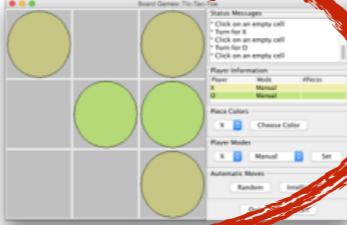
Havannah Swing View



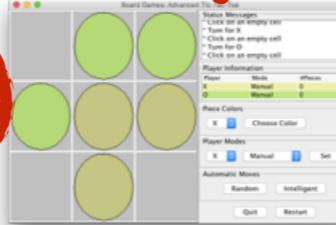
ConnectN Swing View



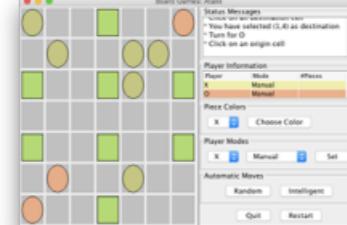
TTT Swing View



ATTT Swing View



Ataxx Swing View



Sólo define cómo el usuario hace movimientos manuales

El tablero se dibuja en RectBoard Swing View

# SwingView - una Posible Disposición

JFrame

colocamos un panel principal con BorderLayout en el “content pane” del JFrame



# SwingView

```
public abstract class SwingView extends JFrame implements GameObserver {  
    private Controller ctrl; // Mantemos una referencia al controlador  
    private Piece localPiece; // Si es null, todos juegan en esta vista. Es importante para decidir si  
    private Piece turn; // desactivamos la vista, hacer un movimiento automático, etc.  
    private Board board; // mantener esta info. actualizada (en los métodos de GameObserver)  
    private List<Piece> pieces; // Los colores de las fichas y los modos de los jugadores  
    private Map<Piece, Color> pieceColors;  
    private Map<Piece, PlayerMode> playerTypes;  
    ...  
  
    public SwingView(Observable<GameObserver> g, Controller c, Piece localPiece, Player randPlayer, Player aiPlayer) {  
        ...  
        initGUI(); // Inicializar la GUI y registrarse como observador ...  
        g.addObserver(this); // Construir la parte gráfica del Control y pedir a la subclase de construir la del tablero (usando initBoardGui)  
    }  
  
    private void initGUI() { ...; initBoardGui(); ... } // Las subclases usan estos métodos para consultar la información correspondiente  
  
    final protected Piece getTurn() { return turn; } // subclases usan estos métodos para consultar y cambiar el color de las fichas ...  
    final protected Board getBoard() { return board; }  
    final protected Board getPieces() { return pieces; }  
    final protected Color getPieceColor(Piece p) { return pieceColors.get(p); }  
    final protected Color setPieceColor(Piece p, Color c) { return pieceColors.put(p,c); }  
    final protected void setBoardArea(JComponent c) { ... } // ... y estos métodos para colocar el componente del tablero, añadir texto a la área de información, etc.  
    final protected void addMsg(String msg) { ... }  
    ...  
  
    final protected void decideMakeManualMove(Player manualPlayer) { ... } // Las subclases usan estos métodos para hacer un movimiento manual ...  
    private void decideMakeAutomaticMove() { ... }  
    ...  
  
    protected abstract void initBoardGui(); // Llamamos a este método para hacer un movimiento automático, si es necesario ...  
    protected abstract void activateBoard();  
    protected abstract void deactivateBoard();  
    protected abstract void redrawBoard();  
  
    // GameObserver Methods  
    public void onGameStart(final Board board, final String gameDesc, final List<Piece> pieces, final Piece turn) ...  
    ...  
}
```

El modelo llama a los métodos del GameObserver para notificar cambios, etc.

# decideMakeAutomaticMove

Este método es muy importante para hacer que el programa juegue automáticamente – no lo usas para los botones de movimiento automático

## ¿Qué Hace?

Si el modo de 'turn' es automático y juega en esta vista, hacemos un movimiento automático llamando a `ctrl.makeMove(p)` donde p es el jugador random o ai (depende del modo de 'turn')

## ¿Cuando Llamarlo?

1. Cuando cambia el turno ...
2. Cuando cambiamos el modo de un jugador de Manual a Automático
3. Cuando empieza el juego (en el caso que mantienes los mismos modos de jugadores – ¡¡recomendable!!)
4. Piensa en que otras partes del programa lo puedes necesitar ...

# Los Métodos de GameObserver

```
public void onGameStart(final Board board, final String gameDesc, final List<Piece> pieces, final Piece turn) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleGameStart(...); }  
    });  
}  
  
public void onGameOver(final Board board, final State state, final Piece winner) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleOnGameOver(...); }  
    });  
}  
  
public void onMoveStart(Board board, Piece turn) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleOnMoveStart(...); }  
    });  
}  
  
public void onMoveEnd(Board board, Piece turn, boolean success) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleOnMoveEnd(...); }  
    });  
}  
  
public void onChangeTurn(Board board, final Piece turn) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleOnChangeTurn(...); }  
    });  
}  
  
public void onError(String msg) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() { handleOnError(...); }  
    });  
}
```

En los métodos de GameObserver, siempre llamar a otro método para hacer algo con la notificación usando invokeLater, así no se bloquea el “bucle de eventos de swing” y tendrás tu práctica ya “thread safe” ...

... además, piensa que hacer en estos métodos, p. ej., llamar a redrawBoard, llamar a decideMakeAutomaticMove, inicializar los colores de las fichas, etc.

# BoardComponent

```
public class BoardComponent extends JComponent {
```

Puede ser JPanel, depende de cómo dibujas el tablero.

```
    private Board board;
```

```
    ...
```

```
    public BoardComponent() {
```

```
        ...
```

```
}
```

```
    public void redraw(Board b) {
```

```
        ...
```

```
}
```

```
    ...
```

```
    protected abstract Color getPiceColor(Piece p);
```

```
    protected abstract boolean isPlayerPiece(Piece p);
```

```
    protected abstract void mouseClicked(int row, int col, int mouseButton);
```

```
    ...
```

desde fuera podemos pedir a este componente de redibujar el tablero, normalmente desde el método redrawBoard

Cuando dibujas el tablero, usa estos métodos para consultar el color o el tipo (jugador o obstáculo) de una ficha. Quien implementa estos métodos se encarga de conectarlos con la información almacenada en SwingView – ver la siguiente página

```
}
```

Cuando se produce un click sobre el tablero, pasa toda la información a este método. Quien lo implementa decide que hacer con este click ...

# RectBoardSwingView

```
public abstract class RectBoardSwingView extends SwingView {  
    private BoardComponent boardComp;  
    ...  
    public RectBoardSwingView(Observable<GameObserver> g, Controller c, Piece localPiece, ...) {  
        super(g, c, localPiece, ...);  
    }  
    @Override  
    protected void initBoardGui() {  
        boardComp = new BoardComponent();  
        ...  
        @Override  
        protected void mouseClicked(int row, int col, int mouseButton) {  
            // call handleMouseClicked to let subclasses handle the event  
        }  
        @Override  
        protected Color getPieceColor(Piece p) {  
            // get the color from the colours table, and if not  
            // available (e.g., for obstacles) set it to have a color  
        };  
        @Override  
        protected boolean isPlayerPiece(Piece p) {  
            // return true if p is a player piece, false if not (e.g., an obstacle)  
        };  
        setBoardArea(boardComp); // install the board in the view  
    }  
    @Override  
    protected void redrawBoard() {  
        // ask boardComp to redraw the board  
    }  
    ...  
    protected abstract void handleMouseClicked(int row, int col, int mouseButton);  
}
```

**Pasar toda la información a la super clase**

Inicializa la parte gráfica del tablero cuando SwingView lo pide: construir el componente y pasarlo a SwingView llamando a setBoardArea

Así se puede conectar los métodos abstractos de la clase BoardComponent con la información almacenada en SwingView, y también pasar los eventos del ratón a la subclase ...

Cuando SwingView nos pide redibujar el tablero lo pedimos al BoardComponent ...

Se usa para pasar los eventos del ratón a la subclase

# ConnectNSwingView

```
public class ConnectNSwingView extends FiniteRectBoardSwipeView {  
    private ConnectNSwingPlayer player;  
    ...  
  
    public ConnectNSwingView(Observable<GameObserver> g, Controller c, Piece localPiece, ...) {  
        super(g, c, localPiece, ...);  
        player = new ConnectNSwingPlayer();  
        ...  
    }  
  
    @Override  
    protected void handleMouseClicked(int row, int col, int mouseButton) {  
        // do nothing if the board is not active  
        ...  
        player.setMove(row, col);  
        decideMakeManualMove(player);  
    }  
  
    @Override  
    protected void activateBoard() {  
        // - declare the board active, so handleMouseClicked accepts moves  
        // - add corresponding message to the status messages indicating  
        //   what to do for making a move, etc.  
    }  
  
    @Override  
    protected void deActivateBoard() {  
        // declare the board inactive, so handleMouseClicked rejects moves  
    }  
}
```

Un jugador que juega  
a lo que se le dice ...

En el caso de connectN es fácil, decimos a player que coloca una ficha en la casilla (row,col) cuando se le piden un movimiento, es decir, el método requestMove devuelvo una instancia de la clase ConnectNMove con (row,col). Después pedimos a SwingView hacer un movimiento manual usando player

El caso de ataxx/attt es más complicado, porque un movimiento se decide mediante varias llamadas a handleMouseClicked, además puede ser llamado para cancelar un movimiento, etc.

SwipeView llama a estos métodos para indicar si se puede aceptar movimientos manuales o no ....

# Conectar los Componentes

```
public class Main {  
    public static void startGame() {  
        Game g = new Game(gameFactory.gameRules());  
        Controller c = null;  
  
        switch (view) {  
            case CONSOLE:  
                ...;  
                break;  
            case WINDOW:  
                c = new Controller(g, pieces);  
                // if we are in multiple view mode:  
                for (Piece p : pieces) {  
                    gameFactory.create SwingView(g, c, p, gameFactory.createRandomPlayer(),  
                        gameFactory.createAIPlayer(aiPlayerAlg));  
                }  
                ...  
                // if we are in single view mode:  
                gameFactory.create SwingView(g, c, null, gameFactory.createRandomPlayer(),  
                    gameFactory.createAIPlayer(aiPlayerAlg));  
                ...  
            default:  
                ...  
        }  
        ...  
    }  
    ...  
}  
... Tienes que implementar el  
método create SwingView  
en las factorías ...
```

Usar Controller en lugar de ConsoleCtrlMVC

En el modo multi-view creamos una ventana para cada jugador

En el modo single-view creamos sólo una ventana para todos los jugadores, por eso el tercer argumento es null.

```
public class ConnectNFactoryExt extends ConnectNFactory {  
    ...  
    @Override  
    public void create SwingView( ... ) {  
        SwingUtilities.invokeLater(new Runnable() {  
            @Override  
            public void run() { new ConnectNSwingView( ... ); }  
        });  
    }  
}
```

# ¡Cuidado con addObserver!

```
public SwingView(Observable<GameObserver> g, ...) {  
    ...  
    initGUI();  
    g.addObserver(this);  
}
```

La vista se registra como observador al final de la constructora. En general esto puede ser peligroso porque puede recibir notificaciones antes de que el objeto sea listo. En nuestro caso esto no va a ocurrir porque usamos invokeLater en todas partes — ¡pero tener cuidado!

```
public class ConnectNFactoryExt extends ConnectNFactory {  
    ...  
    @Override  
    public void createSwingView( ... ) {  
        SwingUtilities.invokeLater(new Runnable() {  
            @Override  
            public void run() {  
                GameObserver o = new ConnectNSwingView( ... );  
                g.addObserver(o);  
            }  
        });  
    }  
}
```

Una alternativa es registrar la vista después de haber construido el objeto en la factoría.