



UNIVERSITÀ DI PISA

TicTacToe

Matteo Pinna

Introduction

Tic-Tac-Toe is a paper-and-pencil game for two players who take turns making the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal row is the winner.

The goal is to implement such game using Java Beans. The system will consist of a graphical dashboard, **TTTBoard**, containing the board, a **TTTController** label, and a restart button, which allows to reset the state of the board at any moment. The board is made of 9 cells, which must be instances of a Java bean called **TTTCell**. In brief, the board constantly checks if the game has ended, either with a tie or a winner, while the controller manages the game flow, ensuring that the two players take proper turns when selecting a cell.

Implementation

The application has been implemented in Java, leveraging the NetBeans IDE. The communication between the three components previously mentioned is entirely event-based through *PropertyChangeListener* and *VetoableChangeListener*.

Each component and its implementation will be further described in the upcoming sections.

Code Style

The project is compliant with the *Oracle Java Code Conventions*¹, to obtain a clean and elegant code base.

TTTCell

The **TTTCell** extends a *JPanel* and consists of two buttons (*JButton*), which are initially labeled X and O respectively. After a player clicks one of the button, the clicked one is highlighted and the other disabled.

¹<https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

The cell maintains a *cellState* property (bound and constrained in the JavaBean's sense), in order to keep track of the situation of that specific cell within the game/board.

In Figure 1 the possible cell states and the transitions between them:

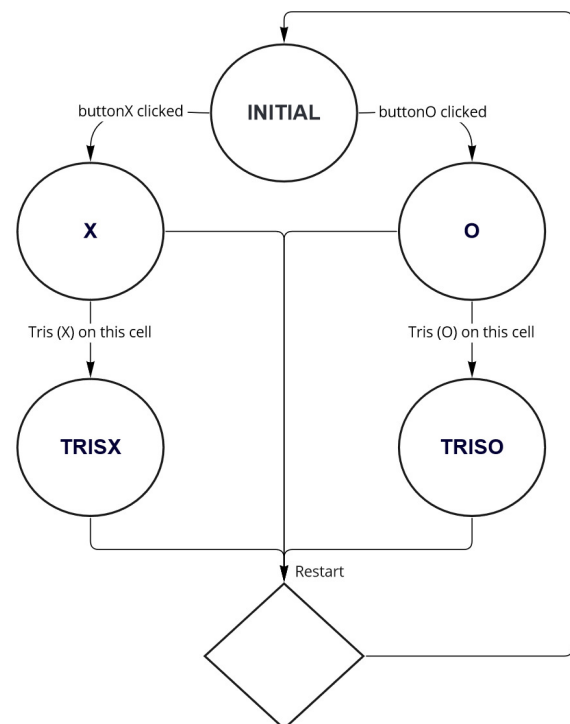


Figure 1: TTTCell's states and transitions.

A cell transitions into either *cellState* X or O when a player clicks the corresponding button, after each player action a *VetoableChange* event is fired to all listeners (*TTTController*) so that the move can be vetoed if the players are not alternating correctly.

The cell must support the following *PropertyChange* events from the board:

- *restart*: fired when the restart button is clicked, the

buttons are reset to their initial state and re-enabled if previously disabled;

- *winner*: fired when there is a winner (i.e. a tris), the cells involved in such tris are highlighted, the other ones disabled.

Notice that each time the cell state changes, a custom background color is set for the buttons.

TTTController

The **TTTController** extends a *JLabel*, communicates the current game state through messages displayed in the label and is responsible for checking that the two players alternate correctly in a game. To this aim, it is registered as a *VetoableChangeListener* to all the cells in the board. The controller maintains a *gameState* property, in order to keep track of the current game situation.

In Figure 2 the possible game states and the transitions between them:

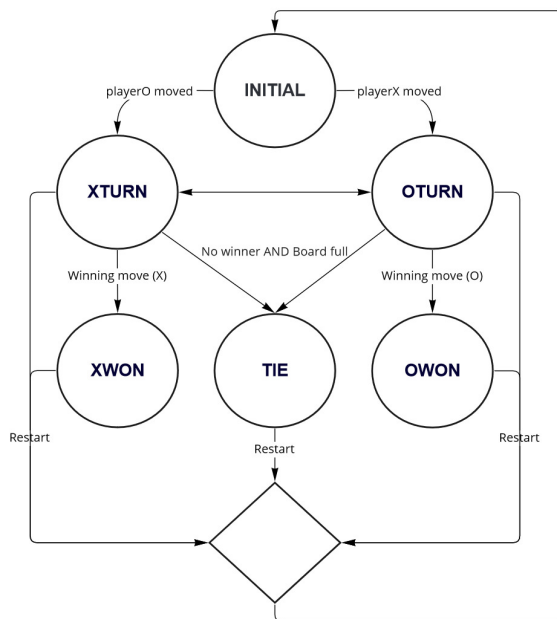


Figure 2: Game's states and transitions.

The controller must support the following *Property-Change* events from the board:

- *restart*: the game state is set to *INITIAL*;
- *tie*: the game state is set to *TIE*;
- *winner*: perform checks to identify the winning player, set the game states to either *XWON* or *OWON*.

Notice that each time the game state changes, a custom background color and text is set for the label.

TTTBoard

The **TTTBoard** is the main class of the application and extends a *JFrame*. It displays a grid of 3x3 *TTTCells*, a restart button and a *TTTController*.

The board is registered as *PropertyChangeListener* to all the cells, so that after each player move it is possible to check whether the game ended with a winner or in a tie, since the board is full and no further move possible. In the first case the winning tris (row, column or diagonal) is highlighted and the label of the controller set to display a corresponding winning message.

The board class is where all listeners are registered:

- Each cell as a *PropertyChangeListener* to the board, in order to receive the *restart* and *winner* events;
- The controller as a *VetoableChangeListener* to all cells, for vetoing invalid moves, and as a *PropertyChangeListener* to the board, in order to receive the *restart*, *tie* and *winner* events;
- The board as a *PropertyChangeListener* to each cell, in order to check if the game has ended after each player move.

The *PropertyChange* events that can be fired by the board have already been discussed in-depth in the dedicated sections of the cell and the controller.

Conclusions

It was possible to develop a simple TicTacToe game leveraging the NetBeans IDE and Java Beans, with interactions between the components entirely event-based.

The source code of the application can be found at this [GitHub repository](https://github.com/NetBeansMP/advanced-programming-assignments/tree/main/first-assignment/1-TicTacToe)².

²<https://github.com/NetBeansMP/advanced-programming-assignments/tree/main/first-assignment/1-TicTacToe>