Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Advanced programming TicTacToe report

Simone Rizzo

562131

Anno Accademico 2021/2022

Contents

1 Introduction **1**

2 Architecture **1**

2.1 TTTCell 1

2.2 TTTController 2

2.3 TTTBoard 3

3 Conclusion **4**

1 Introduction

Tic Tac Toe is a paper-and-pencil game for two players who take turns marking the spaces in a threeby-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. We want to implement Tic Tac Toe using Java Beans. (Note: There are several tutorials on the web showing how you can do it, but none satisfies the requirements listed below.) The system is made of a graphical dashboard, TTTBoard, containing the board, a TTTController label, and a restart button (see Figure 2 below). The board is made of 9 cells, which must be instances of a Java bean called TTTCell. Each cell has therefore the same appearance and the same logic. Conceptually, the TTTBoard dashboard displays the board and checks at each move if the game is ended, while the TTTController label monitors that the two players alternate correctly in a game.

2 Architecture

The three components (TTTBoard, TTTCell, TTTController) interact with each other through the event protocol, in particular through VetoableChangeListener and PropertyChangeListener. Let's now describe how each component works and what event messages sends.

2.1 TTTCell

TTTCell extends JPanel it represents a single cell inside the grid, it has an internal state that can be INIT, X, O according to the transition diagram .

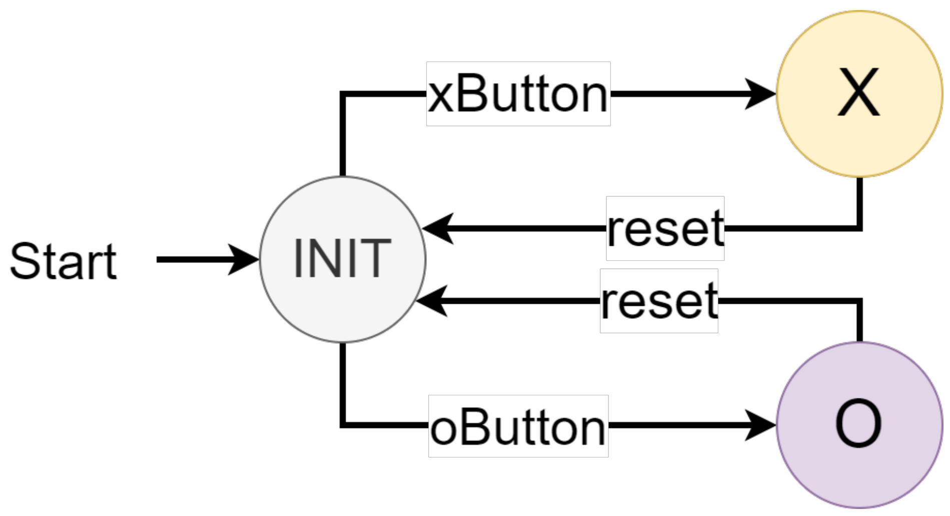

Figure 1: Cell transition diagram

It is composed of two buttons that allow you to set the X or O value. Each click on the buttons launches an event towards the Controller through the VetoableChangeListener in this way if the request is not vetoed then our cell can change state otherwise the operation is canceled.

Each cell is a PropertyChangeListener of the board and vice versa. In this way, every time the cell changes its status, it subsequently communicates with the Board indicating that its status has been updated, in order to allow the Board to calculate if the game is over. If the game is over, the Board sends to the cell which cells have formed the tris. If our cell is one of the winning ones the win function will be called, that set the background color to green. While when the Board send the restart message the cell launch the reset function in order to reinitialize its behaviour.

2.2 TTTController

TTTController extends JLabel, this component manages the game flow taking into account the game turns of the two players through the game state machine that we see in the figure **??**. The game starts with the initial idle state where it waits for one of the players to start the game. It then switches to state O or X and alternates until the game ends with a winner X/O or ends in a draw or is restarted by jump back to idle state.

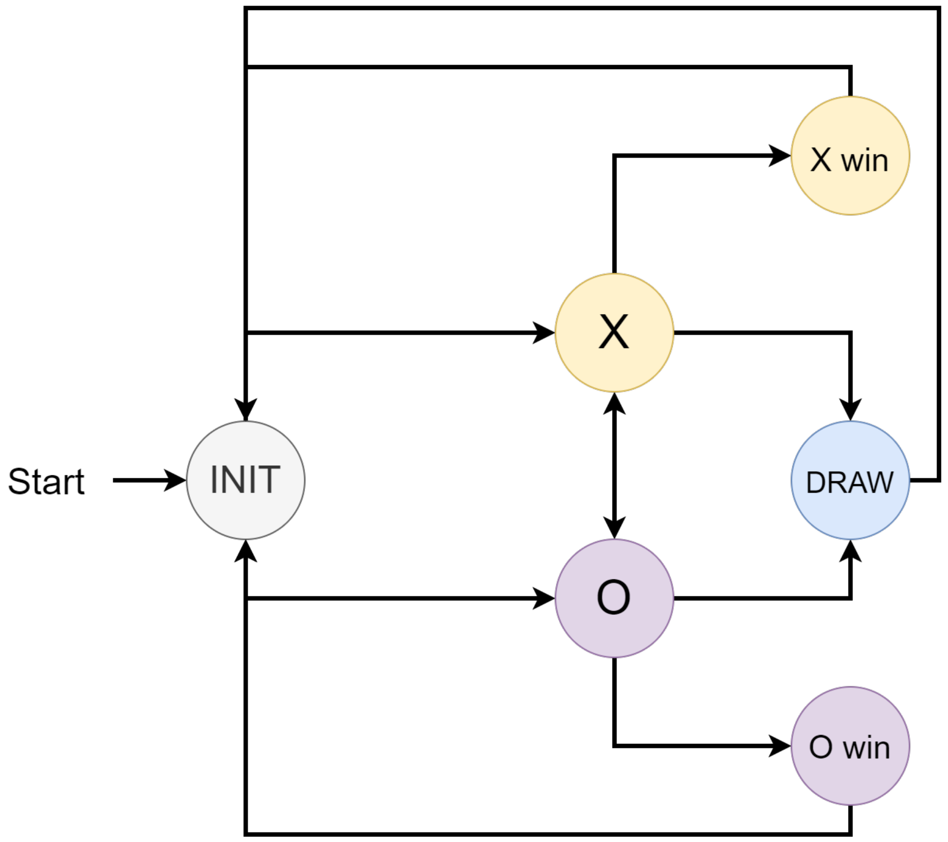

Figure 2: Game state transition diagram

The controller implements VetoableChangeListener to connect it with the cells and check if the actions are allowed otherwise it vetoes by canceling the operation. The Controller is also connected with the Board in order to receive the information about the ending or the restart of the game, it implements two function reset and win. The function reset reinitialize everyting by setting the game state machine to IDLE while win one write the winner in the label and set the backgound to green.

2.3 TTTBoard

TTTBoard extends JFrame and implement PropertyChangeListener, it contains 9 TTT-Cells, the TTTController and a Restart button. Its execution starts by creating a list that contains all the cells and then it initialize and connect the listeners of all the objects. In particular, for each cell it adds the controller as a VetoableChangeListener and the board as a PropertyChangeListener. Subsequently, the board adds each cell and the controller to its own PropertyChangeListener list.

The main task of the board is to analyze when the game is over. To do this, it uses the ProperyChangeListener where it receives the ProperyChangeEvent from the cells that communicate that a cell has been changed. Then check all the possible cases of termination of the game and in case it is finished if the list of winning cells is empty then we are in a case of a tie otherwise it sends a win event by sending the list of cells and a subsequent winner_is event which indicates which player won.

3 Conclusion

In conclusion, the creation of this game allowed us to see the use of Java Components and to make them communicate by using event-base messages. These mechanism were implemented in Java through the use of Listener and Events. Demonstrating how Event-based paradigm with component-based architecture, allows to efficiently design independent and reusable components.