Usability
Undo functionality

In the MVC architecture pattern, the model represents the data and logic of the application, the view represents the user interface, and the controller handles the communication between the model and the view.

Here is the design for the undo functionality in Tic Tac Toe using the MVC architecture pattern:

Model:

The model component in the MVC architecture pattern represents the data and logic of the application. In this case, the model will keep track of the current state of the game board as a 2D array of X, O, or empty values. It will also keep track of the history of moves made in the game as a stack of tuples, where each tuple contains the row and column indices of the move made and the symbol played.

- Keep track of the current state of the game board as a 2D array of X, O, or empty values.
- Keep track of the history of moves made in the game as a stack of tuples, where each tuple contains the row and column indices of the move made and the symbol played.
- Provide methods for updating the game board and adding/removing moves from the history stack.

The model will provide methods for updating the game board and adding/removing moves from the history stack. When a player makes a move, the model will update the game board by placing their symbol in the corresponding cell. It will also add the move to the history stack.

When a player chooses to undo their previous move, the model will remove the last move from the history stack and update the game board accordingly. This means setting the cell to empty and switching the turn to the previous player. If there are no moves left to undo, the model will disallow the undo functionality.

View:

The view component in the MVC architecture pattern represents the user interface. In this case, the view will display the game board as a 3x3 grid of buttons, where each button corresponds to a cell in the board. It will also display a message indicating whose turn it is to play, and a message indicating the winner or a tie if the game is over. The view will also display a button for undoing the previous move. This button will be disabled if there are no moves left to undo.

- Display the game board as a 3x3 grid of buttons, where each button corresponds to a cell in the board.
- Display a message indicating whose turn it is to play.
- Display a message indicating the winner or a tie if the game is over.

- Display a button for undoing the previous move.

Controller:

The controller component in the MVC architecture pattern handles the communication between the model and the view. It initializes both components and sets up the game. When a player clicks on a cell in the game board, the controller will update the model and view accordingly. This means updating the game board with the player's symbol, adding the move to the history stack, and switching to the next player's turn.

- Initialize the model and view.
- Handle user clicks on the game board buttons by updating the model and view.
- Handle clicks on the undo button by removing the last move from the model's history stack and updating the view and game board accordingly.
- Check if undo functionality should be disallowed if there are no more moves to undo.

When a player clicks on the undo button, the controller will remove the last move from the history stack in the model and update the view and game board accordingly. It will also check if there are any more moves left to undo and disable the undo button if there are not.

With this design, the undo functionality can be implemented by simply popping the last move from the history stack in the model and updating the view and game board accordingly. The controller can then check if there are any more moves to undo and disable the undo button if there are not.
Overall, this design separates the concerns of the application into distinct components, making it easier to manage and maintain. By using the MVC architecture pattern, we can ensure that the undo functionality is implemented in a clear and organized way, improving the overall quality of the application.

Code Example Implementation

Below is an example implementation of undo functionality for the Tic Tac Toe game using the provided RowGameGUI class and the MVC architecture pattern

Model:

In the RowGameModel class, we need to add two additional fields: a stack to keep track of all the moves made in the game, and a boolean to indicate if the last move was undone. Here's what the modified class would look like:

```
public class RowGameModel {
    public Block[][] blocksData = new Block[3][3];
    private Stack<Block> moves = new Stack<>(); // added stack to keep track of moves
    private boolean lastMoveWasUndone = false; // added flag to check if last move was undone

    // other code in the class remains the same
    /**
     * Make a move at the given row and column.
     *
     * @param row The row of the move.
     * @param col The column of the move.
     * @param player The player making the move.
     */
    public void move(int row, int col, char player) {
        if (blocksData[row][col].getIsLegalMove()) {
            blocksData[row][col].setContents(player);
            blocksData[row][col].setLegalMove(false);
            moves.push(blocksData[row][col]); // add the move to the stack
            lastMoveWasUndone = false; // reset the flag
        }
    }

    /**
     * Undo the last move made in the game.
     */
    public void undo() {
        if (!moves.empty() && !lastMoveWasUndone) { // check if there is a move to undo and if the
last move was not already undone
            Block lastMove = moves.pop(); // get the last move from the stack
            lastMove.setContents(' '); // undo the move
            lastMove.setLegalMove(true);
            lastMoveWasUndone = true; // set the flag
        }
    }
}
```

Controller:
In the RowGameController class, we need to add a new method to handle the undo functionality. We also need to modify the move method to update the GUI after each move. Here's what the modified class would look like:

```
public class RowGameController {
    private RowGameModel gameModel;
    private RowGameGUI gameView;

    public RowGameController(RowGameModel model, RowGameGUI view) {
        this.gameModel = model;
        this.gameView = view;
    }

    public void move(JButton block) {
        int row = -1, col = -1;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (block == gameView.blocks[i][j]) {
                    row = i;
                    col = j;
                }
            }
        }
        gameModel.move(row, col, gameModel.getPlayerSymbol());
        gameView.updateBlock(gameModel, row, col); // update the GUI after each move
    }

    public void undo() {
        gameModel.undo(); // call the undo method in the model
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                gameView.updateBlock(gameModel, i, j); // update the GUI after undoing the move
            }
        }
    }

    public void resetGame() {
        gameModel.reset();
        gameView.reset();
    }
}
```

View:
In the RowGameGUI class, we need to add a new button to the GUI for the undo functionality, and also add an ActionListener to handle the undo event. Here's an example implementation:

- Add a new JButton to the options JPanel, alongside the Reset button:

```
JButton undo = new JButton("Undo");
options.add(undo);
```

- Add an ActionListener to the undo button to handle the undo event:

```
undo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        controller.undoMove();
    }
});
```