Use Case: User selects style

Step	User's Action	System's Response
1	User selects style	
2		Board will display selected style

Use Case: First User selects X

Step	User's Action	System's Response
1	User presses button on grid	
2		X outputs

Use Case: Second User selects O

Step	User's Action	System's Response
1	User 2 presses button on grid	
2		O outputs

Use Case: Undo button

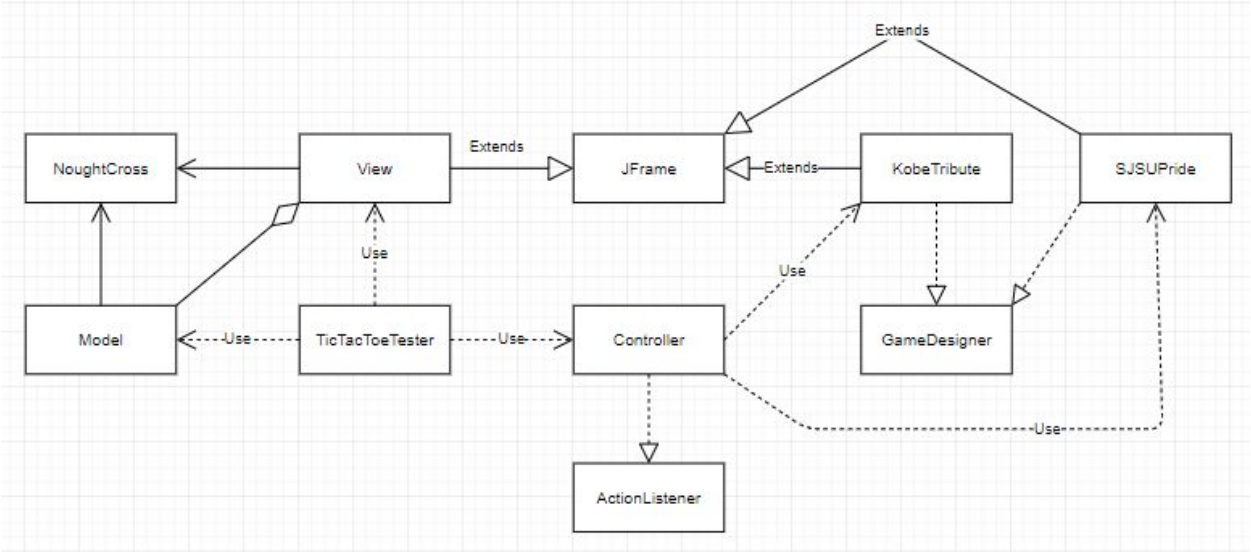
Step	User's Action	System's Response
1	User presses undo	
2		Move gets undone (1 undo used)

3	User plays move (X/O)	
4		X/O gets outputed

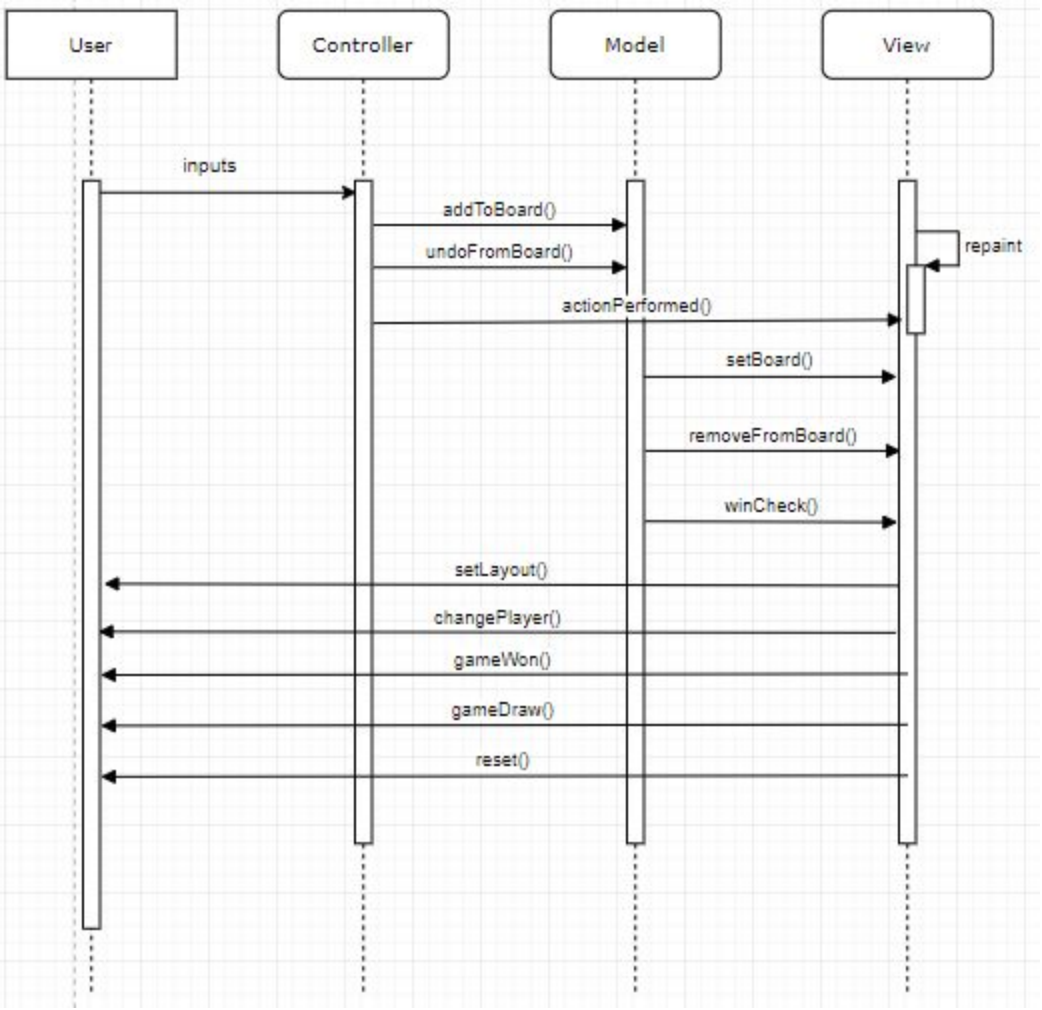
Use Case: Win

Step	User's Action	System's Response
1		Checks best move
2		Check if there are 3 in a row

Simple Class Diagram



Sequence Diagram:



Design Pattern Assessment

Controller: controller.java

```
private boolean addToBoard(Integer field, NoughtCross noughtCross) {
		model.setBoard(field, noughtCross);
		if(Model.totalMoves >= 5) {
			if(model.winCheck()) {
				view.gameWon();
				view.reset();
				return true;
			}
			if(Model.totalMoves == 9) {
				view.gameDraw();
				view.reset();
				return true;
			}
		}
		return false;
	}
	private boolean undoFromBoard(Integer field, NoughtCross noughtCross) {
		model.removeFromBoard(field, noughtCross);
		return true;
	}
```

Model: model.java

```
public void setBoard(Integer field, NoughtCross value) {
		Integer placeVal = null;
		if (value == NoughtCross.O) {
			placeVal = 1;

		} else {
			placeVal = -1;
		}
		gameBoard[field % 3][field / 3] = placeVal;
		totalMoves++;
	}

	public void removeFromBoard(Integer field, NoughtCross value) {
		if(value == NoughtCross.O || value == NoughtCross.X) {
			gameBoard[field % 3][field / 3] = 0;
		}
		totalMoves--;
	}
```

View: View.java (code is from controller class)

```
@Override
	public void actionPerformed(ActionEvent e) {
```

```
NoughtCross currentPlayer = view.getCurrentEnum();


if(e.getSource().equals(view.undo)) {
	undoHelper = 0;
	undoCount--;
	if (undoFromBoard(current, currentPlayer)) {
		view.changePlayer();
	}
	view.square[current].setText("");
	view.undoRemaining.setText(String.valueOf(undoCount));
	view.square[current].setEnabled(true);
	view.undo.setEnabled(false);
}
	else if (Arrays.asList(view.square).contains((JButton) e.getSource()) &&
currentPlayer != null) {
		undoHelper++;
		current =
Arrays.asList(view.square).indexOf((JButton)e.getSource());
		((JButton) e.getSource()).setText(view.getCurrentString());
		((JButton) e.getSource()).setEnabled(false);
		if(!addToBoard(current, currentPlayer)) {
			view.changePlayer();
			if(undoHelper >= 2) {
```

```
                                        undoHelper = 0;
                                        undoCount = 3;

view.undoRemaining.setText(String.valueOf(undoCount));
                                }
                        }
                        if(undoCount > 0 && !model.winCheck() && Model.totalMoves < 9) {
                                view.undo.setEnabled(true);
                        }
                } else if(e.getSource().equals(view.kobe)) {
                        KobeTribute kb = new KobeTribute();
                        kb.changeButton(view.square);
                        kb.changeFrameBackground(this.view);
                }else if(e.getSource().equals(view.sjsu)) {
                        SJSUPride sp = new SJSUPride();
                        sp.changeButton(view.square);
                        sp.changeFrameBackground(this.view);
                }
                else if(e.getSource().equals(view.std)) {
                        this.view.getContentPane().setBackground(null);
                        for (JButton s : view.square) {
                        s.setBackground(null);
                        s.setOpaque(true);
```

```
                    s.setBorderPainted(true);
                    s.setForeground(null);
                    }
            }
            else if(e.getSource().equals(view.reset)) {
                    Model.totalMoves = 0;
                    this.view.dispose();
                    this.view = new View();
                    actionListeners();
                    this.model = new Model();
            }
    }
```

Strategy: GameDesigner.java

```
import javax.swing.*;
/**
 * This class declares methods for designing board of tic tac toe.
 */
public interface GameDesigner {
        // this method changes the color of the background of the frame.
        public void changeFrameBackground(JFrame frame);
        // this method changes the color of buttons.
        public void changeButton(JButton[] button);
}
```

Two Concrete Strategies:

KobeTribute.java

SJSUPride.java

```
 else if(e.getSource().equals(view.kobe)) {
                        KobeTribute kb = new KobeTribute();
                        kb.changeButton(view.square);
                        kb.changeFrameBackground(this.view);
                }else if(e.getSource().equals(view.sjsu)) {
                        SJSUPride sp = new SJSUPride();
                        sp.changeButton(view.square);
                        sp.changeFrameBackground(this.view);
                }
```

5. The majority of the concepts that we used for the project were already taught in class, so we didn't need to do much online research. We were able to find some references online, as well as draw from previous projects that we have worked on.

For the most part, we used topics from chapters 4 and 5. The View class essentially creates the GUI and defines methods that are used in the Controller class in order to update the current state of the GUI with the help of the Model class. The View uses many concepts from Java Swing such as JFrame, JButton, JPanel, and others. The Abstract Window Toolkit was also used in the case of the Container.

The Controller class implements the ActionListener interface and uses these ActionListeners to handle action events whenever a user clicks on the buttons in the GUI. The Controller class makes calls to various mutator methods from the Model class. These methods, winCheck(), setBoard(), and removeFromBoard() are all used to update the GUI.

The Model class defines a few methods that will be used by the Controller class to update the GUI that has been made with the help of the View class. We also used an enum to define the characters of X and O for the tic-tac-toe board. We had to research online how to use these effectively.

We faced a few problems while we were trying to design the JFrame, so that different colors could be used. We were able to change the color of the buttons, but could not change the color of the contentPane and JFrame. Because of this, we researched and found that we have to use the setOpaque method and set it to false. After this, we were able to update the design properly.