**Use Cases:**

Use Case #1: Choosing a theme

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 |  | System displays 2 buttons on the screen, one for the space theme and one for the flower theme. |
| 2 | User selects a theme |  |
| 3 |  | An empty board with a background and element shapes with images corresponding to the selected theme is initialized and shown on the screen. |

Use Case #2: Starting the game with player 1 (“x”)

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | User clicks a square to select the next move |  |
| 2 |  | The symbol (“x”) for player 1 appears on the square that was selected. The undo button becomes enabled. |

Use Case #3: Player wants to undo their turn

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | User clicks their undo button |  |
| 2 |  | If the player’s current undo count is less than 3, the previously selected square is cleared, the undo button becomes disabled, and the current undo count for the current player is incremented by 1. |
| 3 | User clicks a square to select the next move |  |
| 4 |  | The appropriate symbol (“x” for player 1 and “o” for player 2) appears on the square that was selected. The current player’s undo button becomes enabled. The player’s undo count can reach a max of 3 for that turn. |

Use Case #3 Variation #1: User clicks undo button when it is not their turn

1.1 In step 1 player 1 tries to undo after their turn is over

1.2 System doesn’t allow it since their turn is over

Use Case #4: Continuing the game as either player 1 (“x”) or 2 (“o”)

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | User clicks a square to select the next move |  |
| 2 |  | The previous player’s undo button is disabled. The appropriate symbol (“x” for player 1 and “o” for player 2) appears on the square that was selected. The current player’s undo button becomes enabled. |

Use Case #5: Finishing the game with a winner

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | When user get 3 x’s or o’s in a row for either diagonal, row, or column |  |
| 2 |  | System declares the winner for either player 1 or 2, with 3 x’s or o’s in a row diagonal, row, or column |
| 3 |  | System ends the program. |

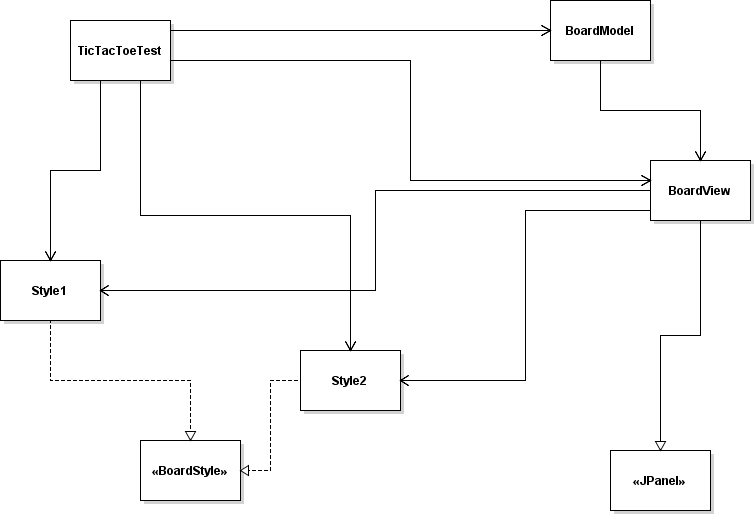
Use Case #6: Finishing the game with a draw

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | When all boxes of the board are filled with no winner |  |
| 2 |  | System declares the game has ended with a draw |
| 3 |  | System ends the program |

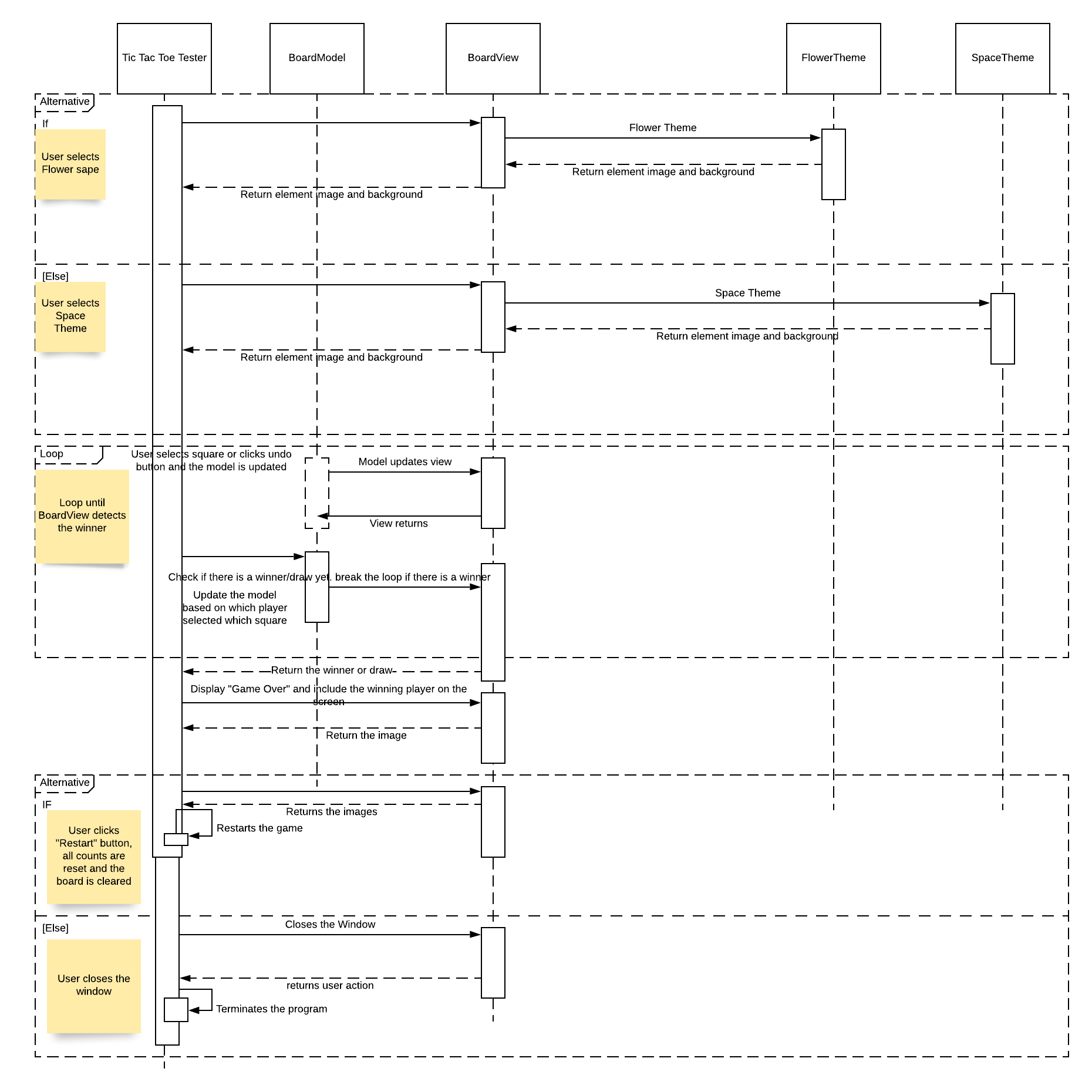
Use Case #7: Starting a new game

|  |  |  |
| --- | --- | --- |
| Steps | User Action | System Response |
| 1 | User clicks the “New Game” button |  |
| 2 |  | All counts are reset to zero and the board and model are cleared. The system displays 2 buttons on the screen, one for the space theme and one for the flower  theme. |

**Class Diagram:**



**Sequence diagram:**



1. Write up for design pattern assessment: (20 points)

Write the NAME of one of the controller classes (or class that contains a controller). Copy and paste a code segment of the controller that calls the mutator of the model.

**Name of controller class: BoardView**

**for (int i = 0; i < 9; i++) {**

**if (a.getSource() == buttons[i]) {**

**if (determinant == 0 && buttonVal[i] == 0) { //If counter is even, it is player 2's icon**

**buttons[i].setIcon(oShape);**

**model.add(i,2);**

**buttons[i].setDisabledIcon(oShape);**

**// buttonVal[i] = 2;**

**} else if (determinant == 1 && buttonVal[i] == 0) { //else, it is plater 1's icon**

**buttons[i].setIcon(xShape);**

**model.add(i,1); //mutator**

**buttons[i].setDisabledIcon(xShape);**

**// buttonVal[i] = 1;**

**}**

**lastIndex = i;**

**buttons[i].setEnabled(false);**

**} else if (a.getSource() == undoButton) { //If undoButton is clicked, reset buttons**

**if (undoCount < 3 && i == 0) {**

**buttons[lastIndex].setIcon(null);**

**model.add(lastIndex,0);**

**// buttonVal[lastIndex] = 0;**

**buttons[lastIndex].setEnabled(true);**

**undoCount++;**

**undoButton.setEnabled(false);**

**alt--;**

**//System.out.print("undo");**

**} else if (undoCount == 3 && i == 0) {**

**counter--;**

**if (alt == counter) {**

**alt--;**

**}**

**}**

**}**

**}**

**Mutator name: add**

Write the NAME of the model class. Copy and paste a code segment of a mutator of the model that modifies data and also notifies view(s). Give me the name of mutator as well.

**Name of model class: BoardModel**

**public void add(int i, int n) {**

**buttonVal[i] = n;**

**}**

Write the NAME of the view class. Copy and paste a code the notification method of the view and show me how the notification method paints the view using the data from the model.

**Name of view class: BoardView**

**public void buttons() {**

**ImageIcon backgroundImage = background;**

**backgroundLabel.setIcon(backgroundImage);**

**backgroundLabel.setLayout(new GridLayout(3, 3));**

**panel3.setLayout(new GridBagLayout());**

**panel3.setLocation(-300, 100);**

**frame2.setSize(600, 700);**

**for (int i = 0; i < model.getButtonVal().length; i++) {**

**buttons[i] = new JButton();**

**buttons[i].getPreferredSize();**

**buttons[i].setBorderPainted(false); //removes border color**

**buttons[i].setContentAreaFilled(false); //this will make the border transparent, by removing the content areas color**

**backgroundLabel.add(buttons[i]);**

**backgroundLabel.add(buttons[i], new GridBagConstraints());**

**buttons[i].addActionListener(this::actionPerformed);**

**}**

**undoButton.addActionListener(this::actionPerformed);**

**frame2.setVisible(true);**

**//frame2.add(view);**

**ImageIcon xImage = xShape; //Set x shape**

**xlabel.setIcon(xImage);**

**ImageIcon oImage = oShape; //Set o shape**

**olabel.setIcon(oImage);**

**panel3.add(backgroundLabel);**

**panel3.add(undoButton);**

**frame2.add(panel3);**

**undoButton.setEnabled(false);**

**}**

Write the NAME of a strategy and copy the code.

**Name of strategy: Board Style**

Write the name of two concrete strategies. (Just names required).

**Two concrete strategies:** **SpaceTheme, FlowerTheme**

Copy and paste the code segment where you create a concrete strategy and plug-in into the context program.

**public class BoardView extends JPanel {**

**private BoardStyle style;**

**public BoardView(BoardModel m, BoardStyle s)**

**style = s;**

1. One page of paper that includes answers for the following questions: (10 points)
   * Which materials/key concepts from this course did you apply on the project?
   * Which topics did you have to learn through self-study in order to complete the project?

On this project, we applied many concepts from the course which made the implementation process go by smoothly. For example, we were tasked with creating and implementing our own interfaces as well as apply different design patterns such as the strategy pattern and MVC pattern. The MVC pattern plays a big part of our program because the game is centered around the view, which takes information from the data model. The data model is edited by the controller method which is found in our view class. In our view class BoardView, the tic tac toe board is displayed using data taken from the BoardData class, which contains the arrays and variables needed. In the BoardView class, the controller method uses actionListener to determine which buttons the user is clicking on and using that data to update the BoardData class and show the results immediately. Furthermore, we applied the strategy pattern to be able to allow the user to select different themes for the tic tac toe board. This was done by creating the strategy as the interface BoardStyle, which declared the methods getBackground(), getXShape(), and getOShape(). All of these methods return an ImageIcon. The concrete strategies we used were SpaceTheme and FlowerTheme, which both implement the BoardStyle interface. These classes implement the methods declared in BoardStyle to return ImageIcon objects corresponding to their own themes (either space or flower theme). In TicTacToeTest, we utilized this strategy pattern to be able to set the appropriate background image, as well as images for the “x” and “o” shapes, on the game screen depending on which theme was chosen by the user on the initial screen. By using both the model view controller architecture and the strategy pattern, we were able to easily organize our code and separate functionalities among different classes in a way to make the code both effective and easily readable.

Although most of the project implementation was straightforward, there were some topics we needed to self-study. Sometimes it is tricky to differentiate between the different design patterns, so further research on the difference between these patterns was needed. For example, the difference between the template method and strategy method as well as the difference between the observer pattern and the MVC pattern, which actually utilizes the observer pattern which is why they seem so similar.