Design Pattern Assessment

Controller Class

BoardBW.java, BoardCW.java

```
public JButton makeButton(int rowIndex, int colIndex) {
        JButton button = new JButton();
        button.setPreferredSize(new Dimension(100, 100));
        button.setIcon(blankIcon);

        button.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    ...
                    bModel.setValue(rowIndex, colIndex, (BoardModel.turnNumber %
2));
                    BoardModel.turnNumber++;
                    ...
                    bModel.setCurrentCoord(rowIndex, colIndex);
              }
        });
        return button;
    }
```

Model Class

BoardModel.java

```
public void setValue(int rowIndex, int colIndex, int value) {
        boardValues[rowIndex][colIndex] = value;
        setChanged();
        notifyObservers();
    }
```

View Class

BoardView.java

```
public void update(Observable arg0, Object arg1) {
       System.out.println("Something changed");
       board.changeIcons(gameButtons, bModel);
    }
```

Board is an object of the concrete method (BoardCW or BoardBW). The method displayed below is from BoardBW:

```
public void changeIcons(JButton[][] buttons, BoardModel model) {
       bModel.checkWin();
       if (bModel.announceWinner().contains("winner") ||
bModel.announceWinner().contains("Player")) {
             for (int row = 0; row < buttons.length; row++) {
                   for (int col = 0; col < buttons[row].length; col++) {
                         if (model.getValue(row, col) == 100) {
                               buttons[row][col].setIcon(blankIcon);
                         } else if (model.getValue(row, col) == 1) {
                               buttons[row][col].setIcon(xIcon);
                         } else if (model.getValue(row, col) == 0) {
                               buttons[row][col].setIcon(oIcon);
                         }
                         buttons[row][col].setEnabled(false);
                   }
             }
             JOptionPane.showMessageDialog(game, bModel.announceWinner());
       } else {
             for (int row = 0; row < buttons.length; row++) {
                   for (int col = 0; col < buttons[row].length; col++) {
                         if (model.getValue(row, col) == 100) {
                               buttons[row][col].setIcon(blankIcon);
                         } else if (model.getValue(row, col) == 1) {
                               buttons[row][col].setIcon(xIcon);
                         } else if (model.getValue(row, col) == 0) {
                               buttons[row][col].setIcon(oIcon);
                         }
                   }
             }
             System.out.println(bModel.announceWinner());
       }
    }
```

Strategy Class

Board.java

```
public interface Board {

       public void createBoard();
       public JButton makeButton(int rowIndex, int colIndex); //draws a blank
button
       public void changeIcons(JButton[][] buttons, BoardModel model);
       public JButton makeUndoButton();
}
```

Concrete Strategies: BoardBW.java and BoardCW.java

```
private void addStrategyButtons() {
       JButton BWButton = new JButton("Black and White");
       JButton CWButton = new JButton("Color");

       BWButton.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                   System.out.println("Black and White"); // remove later
                   board = new BoardBW(bModel); // board is now set to BW mode
                   createGame();
             }
       });

       CWButton.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                   System.out.println("Color");
                   board = new BoardCW(bModel); // board is now set to Color
mode
                   createGame();
             }
       });
       startPanel.add(BWButton);
       startPanel.add(CWButton);
    }
```

“board” is a class variable that references the BoardCW or BoardBW object in the BoardView.java file.

Conclusion

We employed many concepts from this course while writing this project. The first concept was using use-cases, class diagrams, and sequence diagrams to communicate amongst teammates about how the program would be structured. In designing our project itself, we made use of several design patterns. One of them was the Observer pattern, of which we used MVT (Model-View-Controller) structure to control the “data” of our project and how clicking on the buttons altered the data and thus, the visual. We used the Strategy pattern to create two different versions of the board with high code reusability- because we had a Strategy interface, we could easily reference the selected board.

Most of the self-study employed in using this project was primarily for the GUI. Figuring out how to make the board look how we wanted- with a 3 by 3 layout, positioning the components, using pictures on the buttons rather than drawing with shapes- was a new process for many of us. Creating the start screen was also something to figure out, as we were not sure how to change the content of a frame prior to this project.