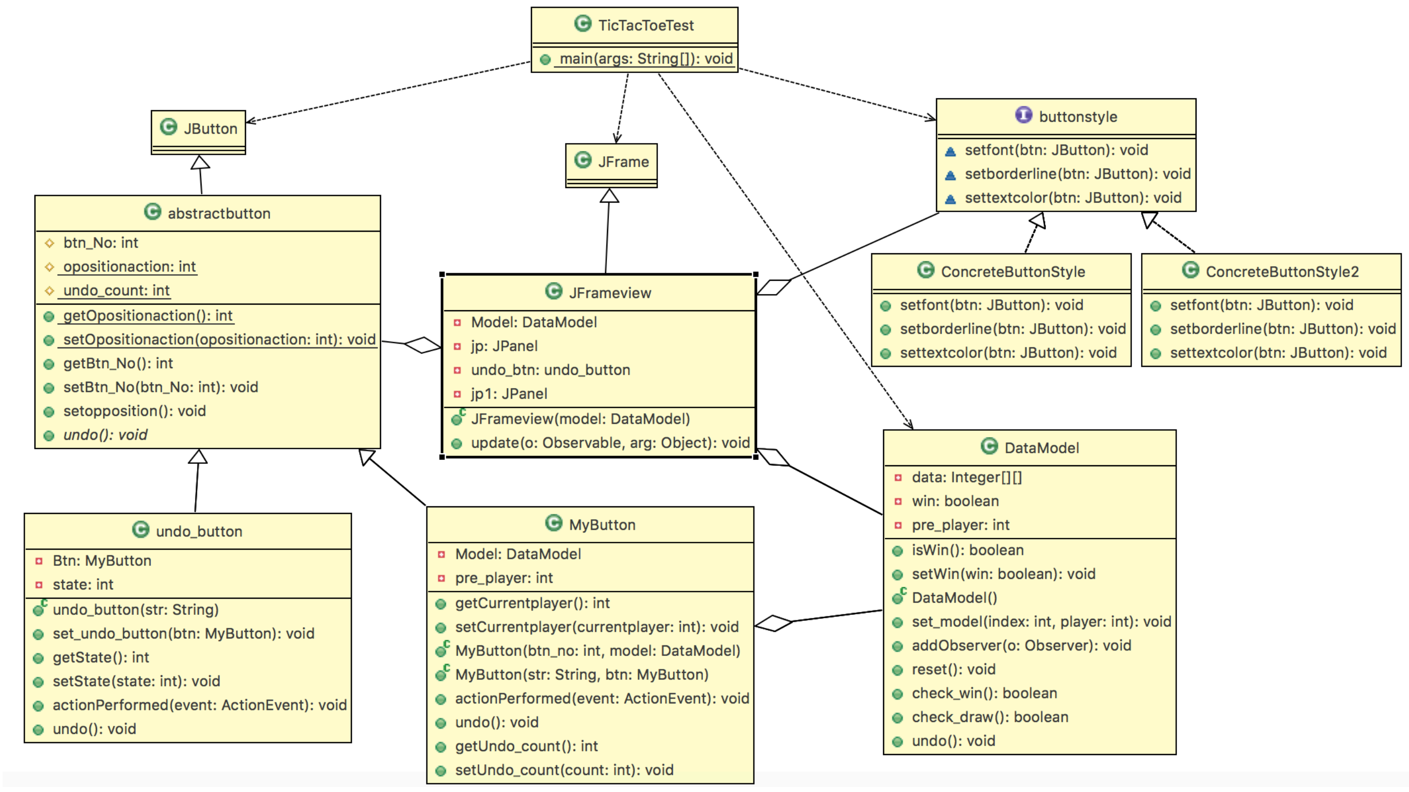
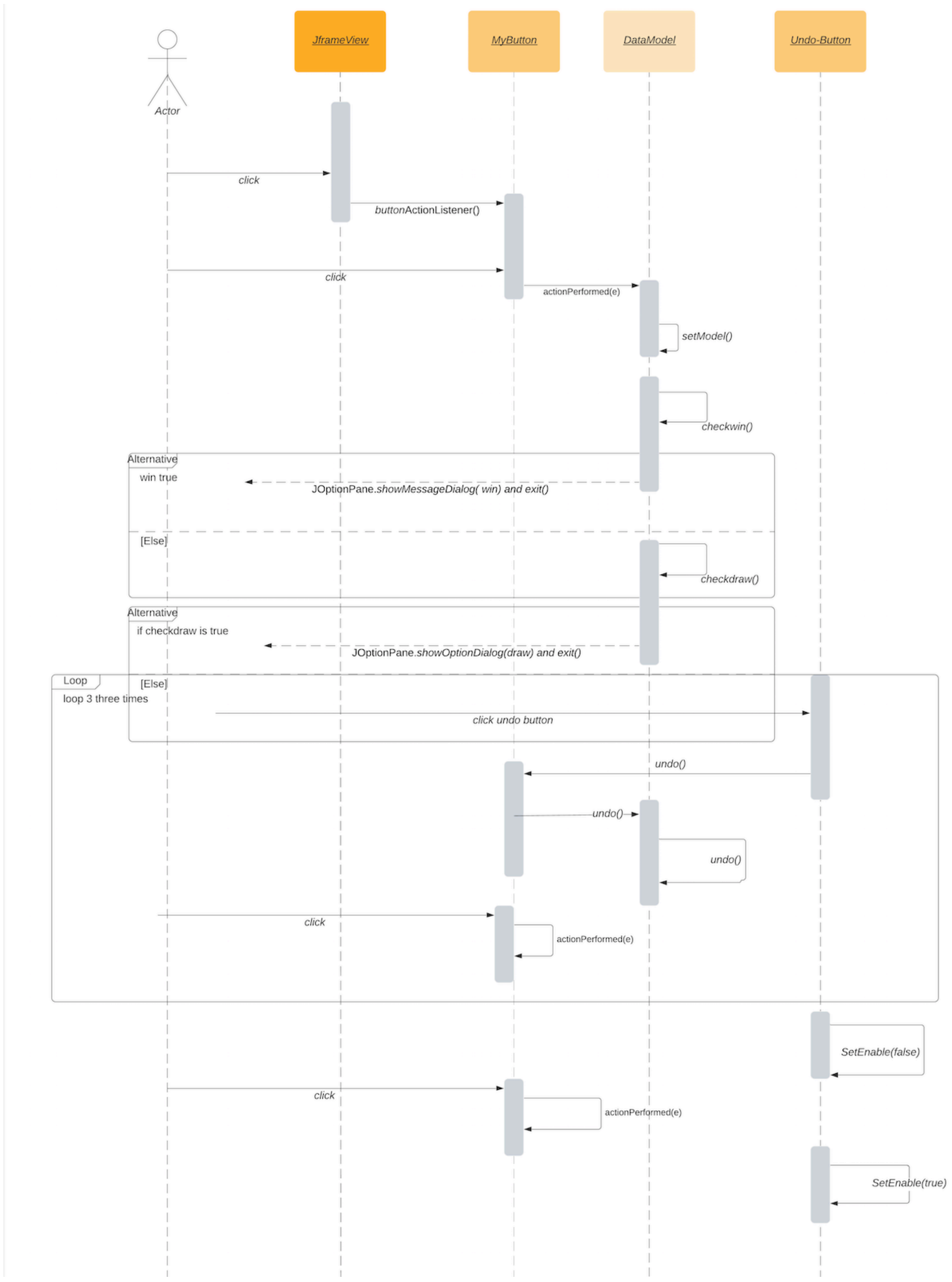
Final Report:

1. Use cases (10 points)

2. Class diagram (simple class diagram - 20 points)


TicTacToeTest
main(args: String[]): void
JButton
JFrame
buttonstyle
setfont(btn: JButton): void
setborderline(btn: JButton): void
settextcolor(btn: JButton): void
abstractbutton
btn_No: int
opositionaction: int
undo_count: int
getOpositionaction(): int
setOpositionaction(opositionaction: int): void
getBtn_No(): int
setBtn_No(btn_No: int): void
setopposition(): void
undo(): void
JFrameview
Model: DataModel
jp: JPanel
undo_btn: undo_button
jp1: JPanel
JFrameview(model: DataModel)
update(o: Observable, arg: Object): void
ConcreteButtonStyle
setfont(btn: JButton): void
setborderline(btn: JButton): void
settextcolor(btn: JButton): void
ConcreteButtonStyle2
setfont(btn: JButton): void
setborderline(btn: JButton): void
settextcolor(btn: JButton): void
DataModel
data: Integer[][]
win: boolean
pre_player: int
isWin(): boolean
setWin(win: boolean): void
DataModel()
set_model(index: int, player: int): void
addObserver(o: Observer): void
reset(): void
check_win(): boolean
check_draw(): boolean
undo(): void
undo_button
Btn: MyButton
state: int
undo_button(str: String)
set_undo_button(btn: MyButton): void
getState(): int
setState(state: int): void
actionPerformed(event: ActionEvent): void
undo(): void
MyButton
Model: DataModel
pre_player: int
getCurrentplayer(): int
setCurrentplayer(currentplayer: int): void
MyButton(btn_no: int, model: DataModel)
MyButton(str: String, btn: MyButton)
actionPerformed(event: ActionEvent): void
undo(): void
getUndo_count(): int
setUndo_count(count: int): void

3. Sequence diagram (20 points)



4. Write up for design patter 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.

MyButton.class

```
public void actionPerformed(ActionEvent event) {
        if (opositionaction == 0) {
            setText("X");
            if (pre_player != opositionaction) {
                pre_player = opositionaction;
                undo_count = 0;
            }

            Model.set_model(btn_No, 0);

        } else {
            setText("O");
            if (pre_player != opositionaction) {
                pre_player = opositionaction;
                undo_count = 0;
            }
            Model.set_model(btn_No, 1);
        }
        setopposition();
        setEnabled(false);
    }
```

• 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.

DataModel.class

```
public void set_model(int index, int player) {
        if (index == 0)
            data[0][0] = player;
        else {
            int row = index / 3;
            int col = index % 3;
            data[row][col] = player;
        }
        pre_player = index;

        if (check_win()) {
```

```
                 System.out.println("notify");
                win = true;
                setChanged();
                notifyObservers();// notify all observer
when changed;
                reset();
            } else if (check_draw()) {
                win = false;
                setChanged();
                notifyObservers();
                reset();
            }

        }
```

• 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.

Jframeview.class

```
@Override
    public void update(Observable o, Object arg) {
        undo_btn.setEnabled(false);
        for (int i = 0; i < jp.getComponents().length;
i++) {

            MyButton b = (MyButton)
            jp.getComponents()[i];
            b.setText("");
            b.setEnabled(false);
        }
        if (Model.isWin()) {
            JOptionPane.showMessageDialog(null, "Win");
            System.exit(0);
        } else {
           JOptionPane.showMessageDialog(null, "draw");
            System.exit(0);
        }


    }
```

• Write the NAME of a strategy and copy the code.
buttonstyle

```
public interface buttonstyle {
    void setfont(JButton btn);

    void setborderline(JButton btn);

    void settextcolor(JButton btn);
}
```

• Write the name of two concrete strategies. (Just names required).
ConcreteButtonStyle and ConcreteButtonStyle2

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

```
JButton style1 = new JButton("style1"); // two buttom
JButton style2 = new JButton("style2");
ActionListener Change = new ActionListener() {
            @Override
public void actionPerformed(ActionEvent e) {
      JButton jb = (JButton) e.getSource();
      if (jb.getText() == "style1") {
         for (int i = 0; i < jp.getComponents().length;
i++) {
    MyButton mb = (MyButton) jp.getComponents()[i];
       buttonstyle style = new ConcreteButtonStyle();
       style.setborderline(mb);
       style.setfont(mb);
       style.settextcolor(mb);
      }

     } else {
  for (int i = 0; i < jp.getComponents().length; i++) {
      buttonstyle style2 = new ConcreteButtonStyle2();
      MyButton mb = (MyButton) jp.getComponents()[i];
      style2.setborderline(mb);
      style2.setfont(mb);
      style2.settextcolor(mb);
      }
     }
```

```
            remove(jp1);
            setLayout(new GridLayout(2, 1, 5, 5));
            jp.setVisible(true);
            }
        };
style1.addActionListener(Change);
style2.addActionListener(Change);
```

5. 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?

1. Observation pattern,
2. strategy pattern,
3. abstract class,
4. interface,
5. inheritance
6. static instance variable
7. swing GUI

• Which topics did you have to learn through self-study in order to complete the project?

1. decorate the button part.
2. Command pattern