Final Report: Group 14

Aadit Kapoor

Zefeng Qiu

Sandeep Isher

San Jose State University

CS 151

Tejaswini Karra

https://github.com/aaditkapoor/tic-tac-toe-cs151

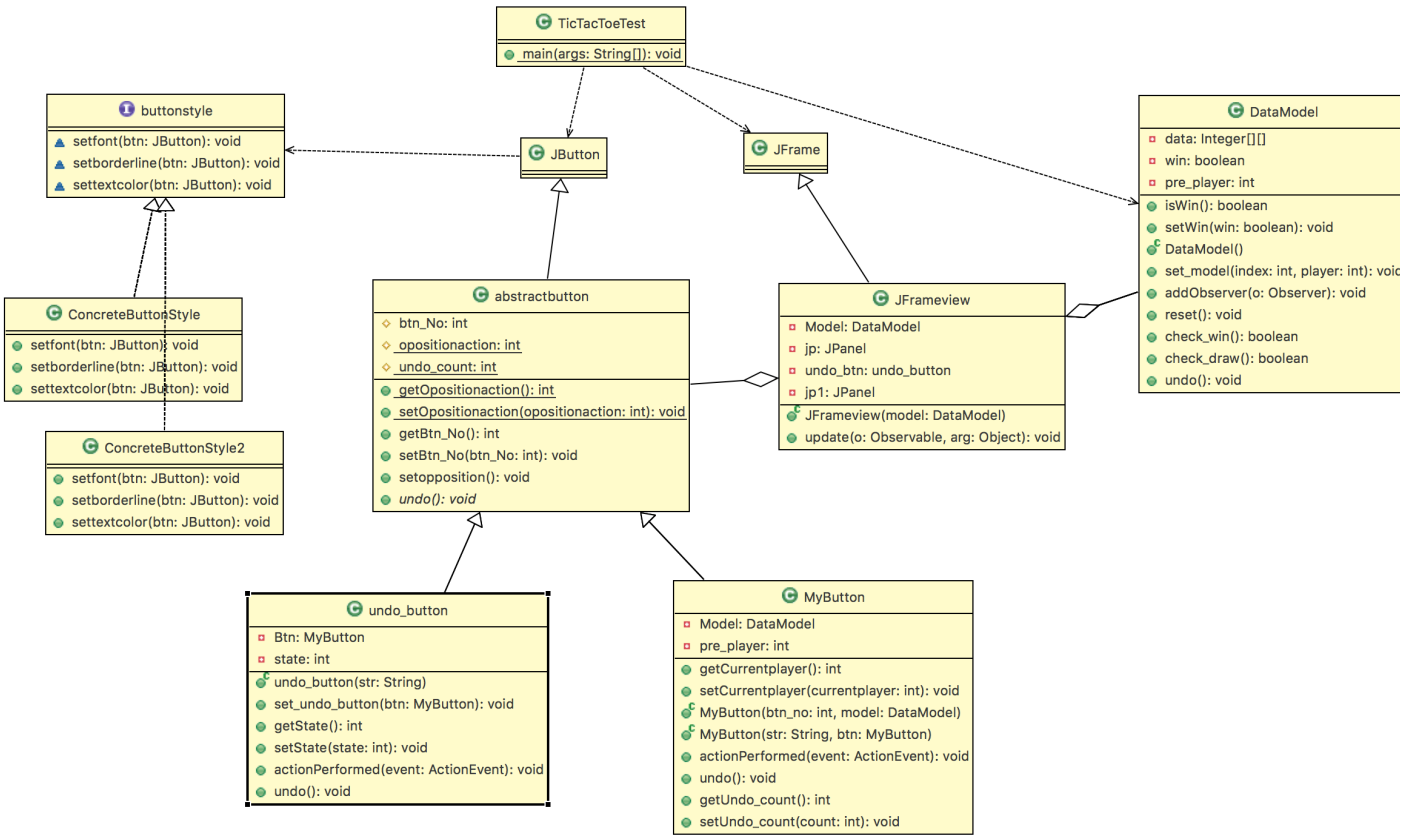
Use cases (10 points)

Steps	User's Action	System's Response
1	User starts Tic-Tac-Toe game	
2		System ask to select board style (from two styles).
3	User selects board style.	
4		System selects style and directs to the playground according to the style selected
5	First game player places a X on an open space on the game board	
6		System marks that space at occupy on the playground. Gives the player the option to undo its turn as long the other play did not make a turn.
7	Second player places a O on an open space on the game board	
8		System marks that space at occupy on the playground. Gives the player the option to undo as long the other play did not make a turn. And a maximum of 2 times during a game.
	Go back to step 7 as long any of the player does not have 3 in a row or all 9 spaces are occupied	
9		System response player one/two is the winner.
10		System quits the game

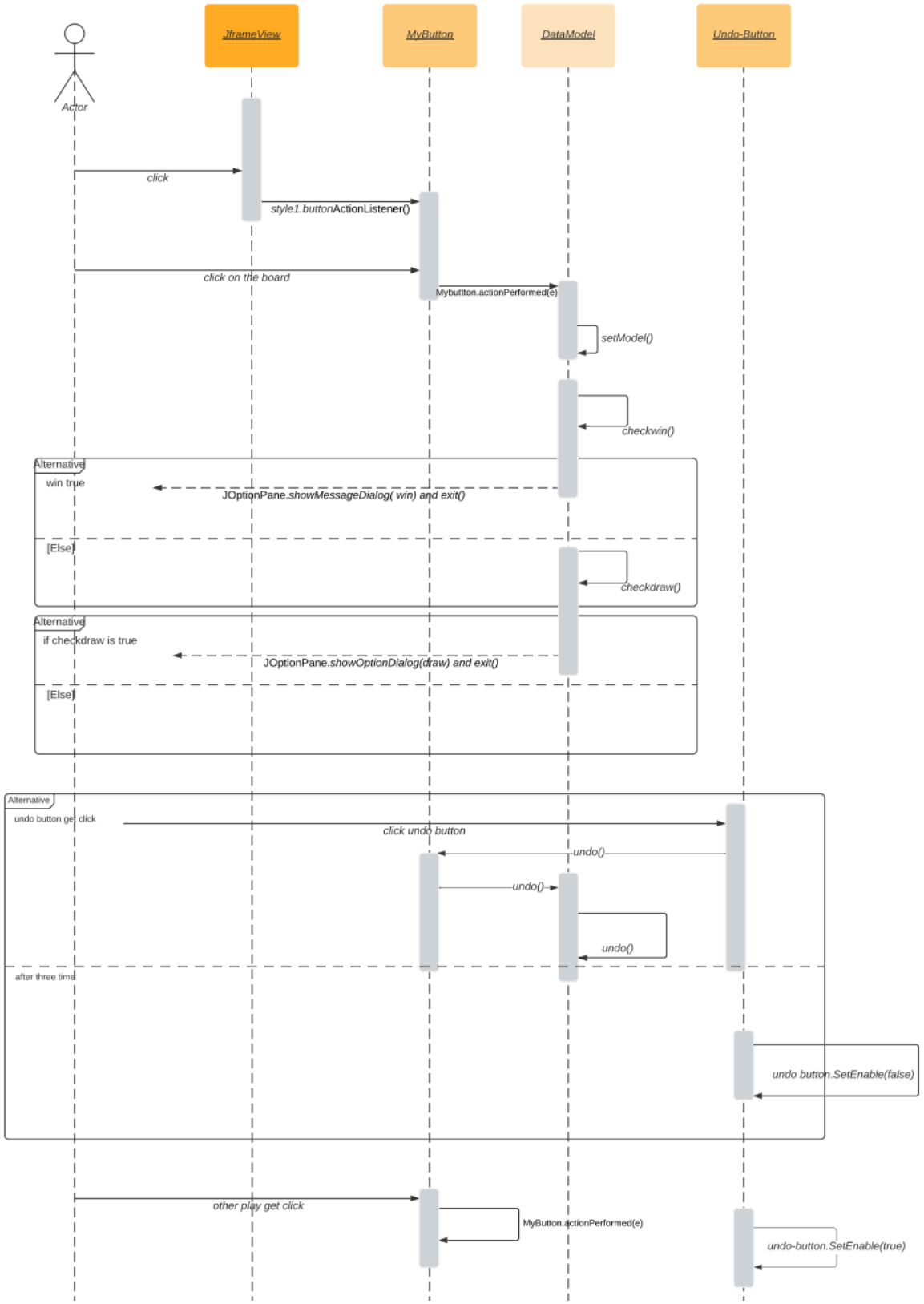
Variation #1Player chooses to undo its turn
2.1.1 Start at step 8 or 10 respectively (User choose to undo)
2.2 System allows player to select another square
2.3 Continue with step 7 or 9 respectively

Variation #2 Player chooses occupied space
3.1 Start at step 9 or respectively at any turn but first turn, when Player choose occupied space
3.2 System does not allow to choose that field
3.3 continue at step 9 or respectively where player choose occupied space

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



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?

The primary premise of this project was to apply Object Oriented Design in our codebase. Design Patterns are an essential part of software engineering and being successful at them is more of an art them a concept to learn. Using patterns in our Tic Tac Toe project, we were able to see how easy is to structure and maintain our codebase.

For this project, we applied an MVC (Model View Controller) pattern or fundamentally we applied the Observer, Strategy, Abstract Pattern. Our layouts (or styles) are managed through the strategy pattern while our MVC is typically based on the Observer pattern. We also heavily employ the use of interfaces, inheritances and certain important Object-Oriented Programming concepts.

During the course of this project, we also attained a reasonable knowledge of the Swing API where we played around with different buttons, layouts etc.

We also employed the use of UML Diagrams to better our workflow and classes.

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

During the course of the project, we tried to make our codebase as structured as possible by trying out separate patterns for certain parts of our functionality like builder pattern and command pattern. We saw that using Builder pattern just made our codebase a lot bigger, but it did help us create objects from complex functionality. We also used the Command Pattern to achieve certain aspects of our project by encapsulating the information needed by an object to perform a certain event.

The project also allowed us to learn about collaborative project building as we all used an industry standard version control system. (git).

The project also allowed us to expand our knowledge about software engineering and we also stumbled upon how we can build an algorithm that can play Tic Tac Toe with a user. (Minimax Algorithm) which we will put in the future most probably bundled with a full-fledged backend system written in Java or something other scripting language. (Python, JavaScript).

To conclude, building this project has been a wonderful journey and we thank Professor Kara for this opportunity.