6.170 Assignment 4 Documentation

Dina Betser

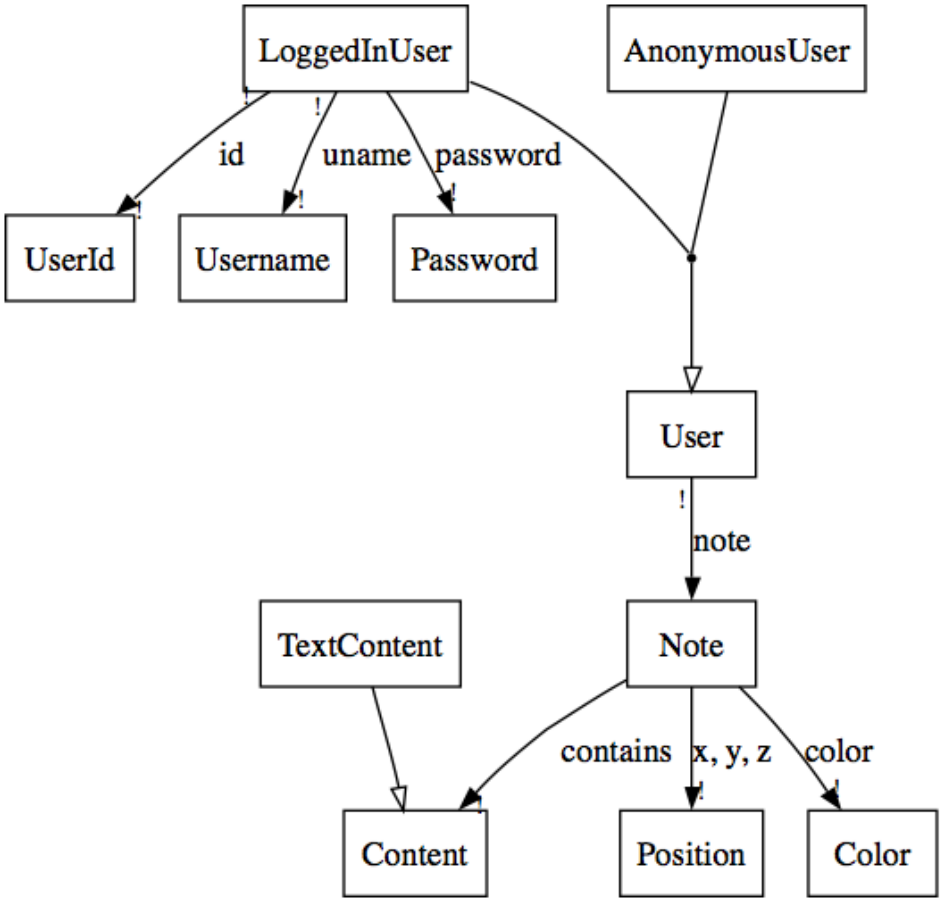
April 1, 2012

1 Models

1.1 Object Models

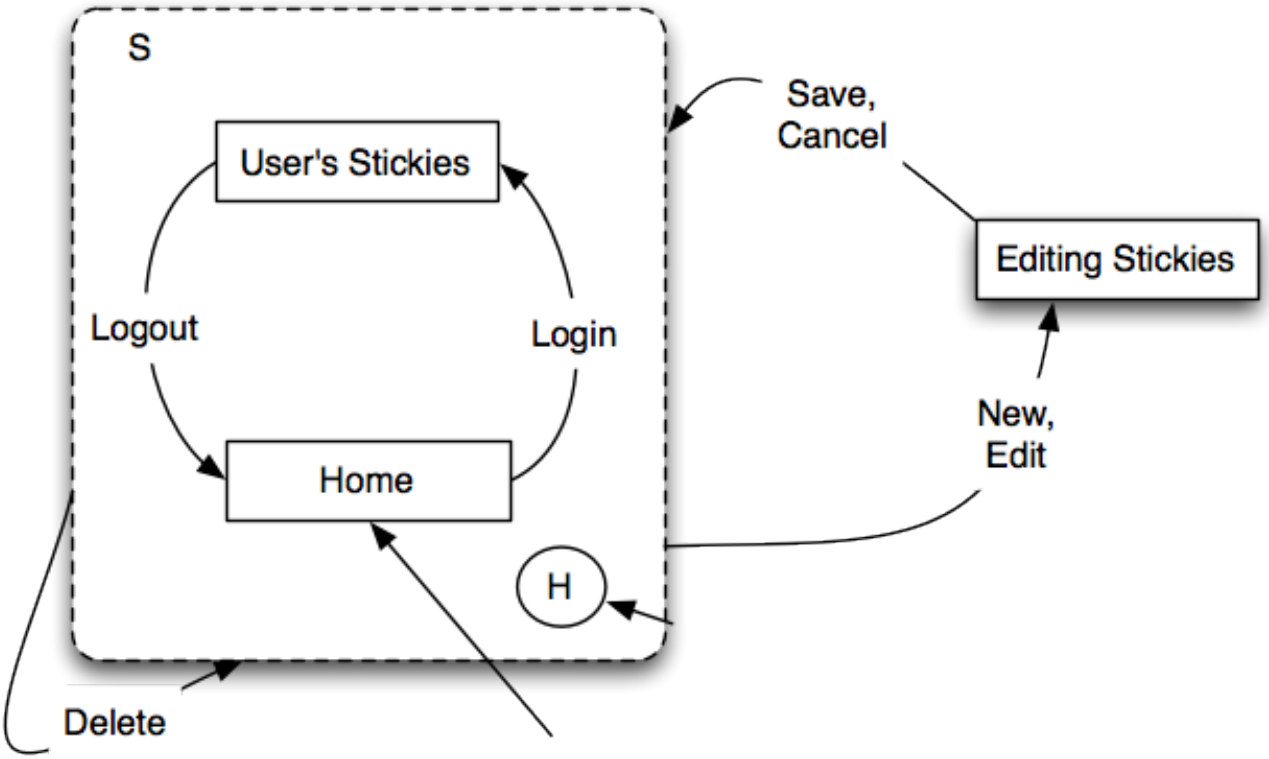
The object model for the problem domain is included in the figure below. The problem domain object model demonstrates the system that must be built.

The top-level object is a `Game`, which includes the concepts of `Players`, a `Board`, a play `Mode`, a `GameStateHistory` for undo/redoing moves, and `GameStats` to store things like the number of boxes allocated to each player.



1.2 State Machines

The following state machine describes how the state of the application changes as game play progresses.



2 Design Notes

2.1 Key Challenges

- Displaying/deleting notes
- Does a user need to be logged in to create stickies?
- Editing notes on the fly or using a popup dialog.
- Storing the state in the frontend for a dirty sticky.
- Save all "current stickies" on login.
- Status of server – should it be stateless?

2.2 Issues Arising

- Implementing login – using flask-login module or not.
  too buggy
- Using ajax for adding stickies

2.3 Critique

This project implemented separation of concerns by using the MVC design pattern. Each module created had a clear specification and purpose, so the code itself was organized fairly well. All view/controller code was stored in `reversi.js`, while all model code was stored in `reversi_model.js`. All code relating to updating the user interface was called from the view or controller; the model always called these view updates using callback function passed in to the model.

The `GameStateData` data structure was particularly succinct at storing all of the information required to store and restore the state of a game during undo/redo.

The implementation chose more sophisticated notions of undo/redo than required; for instance, one way to view "undo"ing is that before a user submits a selected box, he can switch his selected box. In addition to implementing that form of undo, I also implemented a chain-based undo where previous states of the board could be restored from any given state.

The implementation also allowed a user to change play mode in the middle of the game, something that many implementations for this project did not.

3 Specification

3.1 Overview

This application is a web application written using HTML/CSS/JavaScript on the client-side, and the Python Flask framework (with shelve for persistent storage) on the server side.

In addition to an implementation of the general game logic, the project included an interactive game board that allowed up to two human players to play the game. The application allows users to undo and redo moves such that previous states of the board are restored, and alerts the users when the game has finished, displaying who won the game. Two modes of play are supported and a game can be aborted/restarted at any point during game play.

3.2 Key Features

The key features of this implementation of the 6.170 Network Stickies implementation are:

- Ability for users to access stickies from any browser.
- Smooth dragging feature to arrange stickies anywhere in the field of view.
-

3.3 User Manual

The running application can be accessed at http://web.mit.edu/ dbetser/www/6.170/assignment3/reversi.html.

The desired type of play should be selected from the dropdown menu.

At any point, the "Restart" button can be clicked in order to return to the initial board state and start afresh. This effectively aborts the current game and starts one from fresh.

With 1-player play, the black player begins. The desired box is selected by clicking, whereupon it is highlighted in purple. Only valid moves can be selected in this way. The selected box can be changed until the user is ready to submit the move, at which point the "Submit" button should be clicked, or the "enter" key pressed.

Upon submitting the move, the turn switches to computer, which places a marker, switching play back to the human player. With 2-player mode, the turn instead switches to the white player, who is also human. At any point, moves can be undone and redone as long as there are moves to undo/redo.

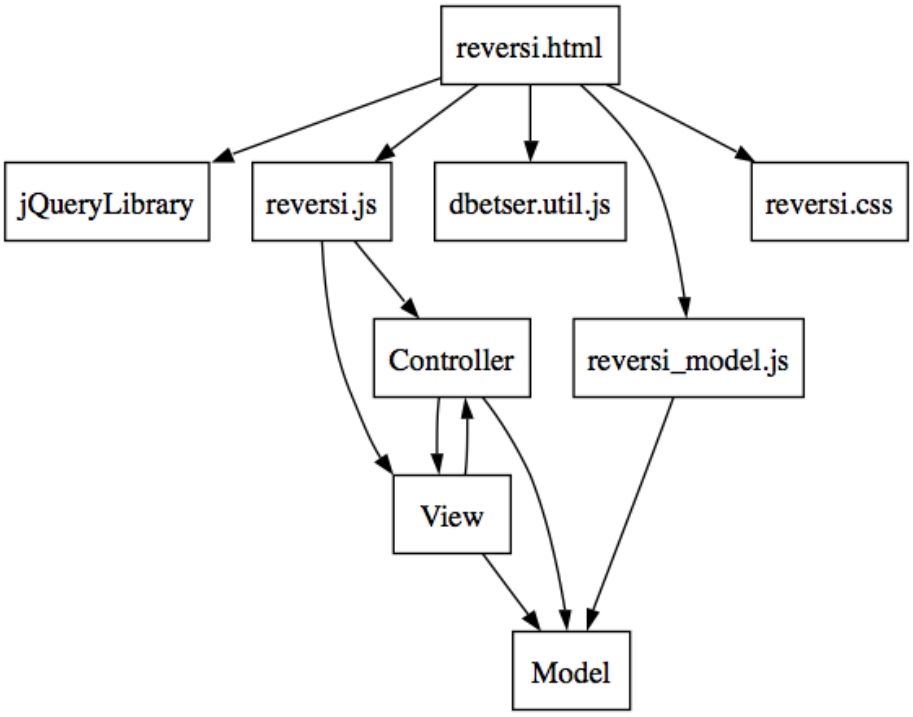
When the game ends, either because all boxes are occupied or no more valid moves are possible, the outcome is displayed to the user beneath the board.

4 Implementation

4.1 Module Dependency Diagram

This code's modules can be seen in the context of the Model-View-Controller framework, which is roughly related to how the files were defined. The `reversi.js` file contains the View and Controller, while `reversi_model.js` contains the model, which includes the `Game` ADT as well as the `GameStateData` ADT. Concerns are separated cleanly in that no code in the model touches anything in the user interface. Whenever the model needs to communicate a change in state to the View, a callback function is used such that a function in the view refreshes the View state. The Controller consists of functions that handle events in the UI, such as the hover and click handling functions.

This code is written using the jQuery library for DOM handling and convenience functions. The figure below shows the module dependency diagram.



4.2 Code Notes

The most major hack that was included in the project deals with updating the player whose turn it is during undo/redo moves. One of the problems with my implementation was that for the two different types of play (versus computer and versus human), the current player during undo needed to be switched for versus human play and needed to stay the same for versus computer play. I included a hack to manually switch the current player in the `getCurrentBoardState` function based on the type of play. This allowed the current player to be correctly displayed even after a series of undos and redos.

Another note is regarding the representation of markers in the UI. In order to present a box's state, I wanted to be able to include a round circle representing the marker that would be used in a real Othello game. However, to do this, I needed to figure out how to change the CSS to include the image of the circle. Using CSS sprites seemed to do the trick for that.

5 Testing

5.1 Test Plan

To test the application, the HTML was validated to ensure standards compliance. Lint was run to avoid poor programming constructs.

The testing for this project was mostly manual. The below was done in both the Firefox and

Chrome browsers.

I developed a number of test cases that ensured that the game functioned as specified.

5.2 Test Cases

To test the board itself, I did the following:

- Play the game until both players have won, in the versus computer and versus human modes. Ensure that the outcome message is displayed appropriately in all cases (black wins, white wins, tie).
- Ensure that refreshing resets all state during any other time of game play.
- Ensure that once a first box is selected, another box can be selected without affecting the model.
- Test undo/redo with both modes to ensure that the board, statistics, and hovering marker change accordingly.
- Test various combinations of undo/redo to ensure that no bug occurs when restoring state.

5.3 Rationale and Conclusions

This project fulfills the requirements of the assignment. The application runs as desired, and even implements undo/redo in a sophisticated and extensible way. Because I used extra slack days, I was able to debug to create a much cleaner and smoother product than I would have otherwise!

Many of the design decisions implemented made the interface easier and cleaner to use, which improved the overall user experience.