Brian Gouldsberry

6/1/2016

**Project 4: Board Game Framework**

**Design Overview**

The design of the Board Game Framework consists of four main items, the Board Spot, Game Controller, Game Model, and Game View classes. To start off, the Board Spot describes the most basic component of a board game, the spot. It uses the strategy pattern to allow for a different amount of spot types and representations. It is up to the plugin to determine what these values and representations should be, therefore they should include an enum that implements this file. The Board Spot class allows developers to implement their own unique strategy on how to display the board, both using GUI (with help from the Game Renderer) and the console. Console representations of spots are fairly simple and come straight from the toAscii function so each enumeration value can have its own ascii representation on the console. However the GUI version is more complicated. Using reflection, the Game Renderer finds the possible values for a Board Spot in the specific game and then attempts to match them to an image icon that the developer sets inside of the images directory. The renderer then sets the image for each cell in the GUI to whatever image matches the enumeration value for that spot in the board.

Next we have the Game Controller that acts as the controller for the Game Model with the help of the plugin’s own listener class that deals with more specific and plugin unique actions. The controller and listener create one of the three components of the model-view-controller used in the framework along with the Game Model and Game View. Its main purpose is to deal with mouse clicks performed by the GUI. For each click, the controller checks to see if there is a JTable that is set to it, then figures out which cell was clicked and tells the model to operate on the cell. The listener doesn’t exist in the framework, but rather is created by the developer of the plugin. Its job is to deal with specialized tasks that the game must perform based upon user interaction. These include all of the INI preference options that the plugin can perform that are non-normal to normal Game Models. So if a plugin wanted to have a menu option to invert all of the board enums, they could edit the INI file, have a listener for that action command, and then pass the work to the model.

The third component is the Game Model, which assists on being the observable object for the game view observer. The model itself is fairly basic and only holds a status string along with a more complicated Game Table Model. The model notifies its observers when it has been changed so that data from its model and the status can be displayed on the Game View. The Table Model is a more interesting class actually. It composites Board Spots to represent the game board as a group of enumeration values. This allows us to abstract the game board into basic values and be able to modify the board based upon which cell the user clicks on. The Table Model itself is also an Observable for the Game GUI to automatically update the JTable when the two dimensional array of Board Spots has been modified. This happens easily with the GUI but is slightly more difficult to do with the console. In the console, the Game Model must work like more of a traditional Observable by allowing the console to directly access the Board array and print its contents to the console when the update method is called.

Finally the Game View utilizes both the Strategy and Observer pattern to visually display the information to users. Developers have the option to choose between two strategies, a GIU or a Console view. The Game GUI uses a special Game Table as a JTable to display the game board and allows for an image to be set as the background of the table. A Piece Renderer is also used on the table to link board spots to specific images that are linked through the enumeration name and files found in the images directory. The GUI uses a combination of the Game Model and the Game Table Model to update its status label and table respectively. The console uses much lower technology and rather simply prints out ascii representations of each spot in the model’s board array along with the status every time its updated; The user chooses between these two strategies by setting a flag in a command line argument upon starting the program. The flag determines which one is used as the games view.

**How-To**

Below are the steps to implementing your own game with the framework

1. Create a new package in source that is the name of your game in lowercase
2. Create three new classes in your package: [GameName]TableModel, [GameName]Enum, and [GameName]Listener
3. For your TableModel
   1. Have it extend GameTableModel
   2. Implement all methods from GameTableModel
   3. SetSize should set the size of the gameBoard
   4. ValidateMove should be the crux of your game; it should deal with what happens when a piece is selected to be moved. It should modify the board array that is inherited from the framework and then call the fireTableDataChanged() method. This method should also return a String to be shown in the UI to update the user on the move.
   5. Implement more public methods if you plan on adding additional functionality via preferences
4. For your Enum
   1. Make it an enum rather than a classs
   2. The enum should contain values for every possible graphical situation that a cell can be in. For example a checkers game may have the values BLACK, RED,NONE
   3. In the toAscii method you should return a string value that you would like to represent specific enum values. For example return “R” when the enum value is RED
5. For your Listener
   1. The listener only has to have one method, the actionPerformed method, however it does a lot of work
   2. In action performed you should check for every action command that could possibly be called.
   3. “Quit” and “New Game” are default actions and should be accounted for
   4. Add cases in this method for every item that you put in the preferences menu, ie. If you have a submenu Color with an option Red, then you should look for the action command “Color-Red”
   5. When dealing with each action command you can either use built in functions to the model like setTheme, or you can cast the TableModel to your game’s version of it and call additional public methods that you have created
6. To add preferences
   1. Create a new file called [GameName].ini in the config directory
   2. For each desired Submenu of the preferences menu add a [] tag to the ini file that includes the submenu name inside of the brackets
   3. For each desired option for a submenu then add a line in the format “menuitem = menuitem”
      1. The value after the equal sign will not be used so don’t worry about its value
   4. This will link each item to an action in your listener based upon a command in the form of “submenu-item”
7. Add images and background for the GUI
   1. In the images directory make a directory with the name of your game and then inside that make a directory called “Default”
   2. Inside default put in your desired images with in JPG form
   3. The name of your background image must be “background.jpg”
   4. The image to be shown for each specific enum must be named with the enums toString value and extended with .jpg
   5. You can add additional themes by repeating this process but putting the files in a new directory that isn’t “Default”
8. Your game is now ready to play, you can start it up by cleaning and building the project and then using the command java -jar dist/BoardGameFramework.jar [GameName] –g in the base directory of the project
   1. You can run the program using the console view by replacing –g with the –c command

**Known Issues**

1. The undo feature on pawns correctly sets the board back but doesn’t remove any points that resulted from the previous move
2. The game determines a win by elimination based upon comparing the score to the expected amount of enemy pawns. This makes it possible to win the game by taking advantage of the previous issue
3. The framework allows users to make moves after the game has finished
4. In the console, an invalid preferences input will lead to an exception being thrown