The goal of this part of the project was to give users ability to game pieces around without the actual logic of the game applied. We can look at the user interface as a Model View Controller. MVC is an architectural pattern that separates the logic from other components. In this case, chess logic is separated from the user interface and the server. This part of the project (the UI) covers view component which will allow the user to interact with the pieces and the displays the list of moves made.

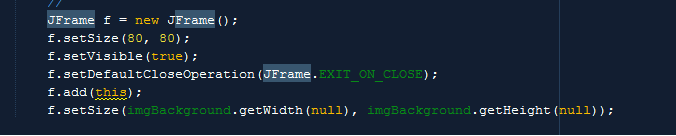
While the control covered the inputs received by the player, the modes implemented the logic of the game and its current states.

**Chess GUI**

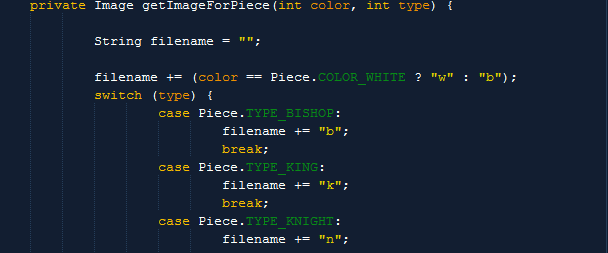
Initially, the program contained a nested for loop that created the board and the background colors were set according to chessboard representation. But this turned out to be even a bigger challenge and more tedious. So instead we used a background image that already contained the squares. We kept track of the game pieces location by storing the pieces in a list setting constant that allowed us to adjust the position of the piece in an location(x,y) format.



Keeping track of this X,Y coordinate is a vital part on the GUI logic. This piece is what will allow us to merge with the logic of the game and let us know whether the move being made is legal.



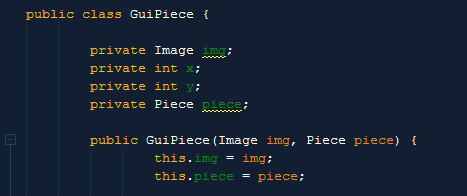
This always has to be called from the main because when you run the program, this is what makes the window appear.

This part loads particular image based on color and type of piece. 

**GUIPieces**

This class simply represented the game pieces and its corresponding x,y coordinates that’s used to display the game pieces.

Code snippet:



**Event Listener**

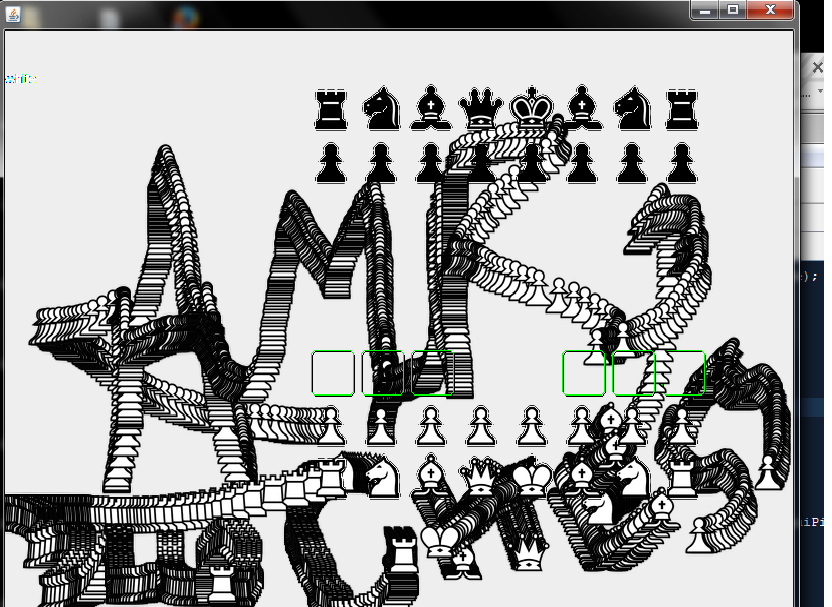
This was the last piece of the GUI phase. Drag and Drop listener implemented events associated with mouse, mouse listener and motion listener in particular. This design was chosen simply out of convenience. We could have used Clicked and release event just as easily instead of implementing drag and dropped. In some cases it would have been easier to implement click and release event. I.e. you click on the piece and click on the square that you want the piece to move to. If the square is a legal zone then drop the piece there, if not. (if it’s a gray zone) nothing happens. Below is drag and drop listener snippet.

simple code sample on how to create Jframe 

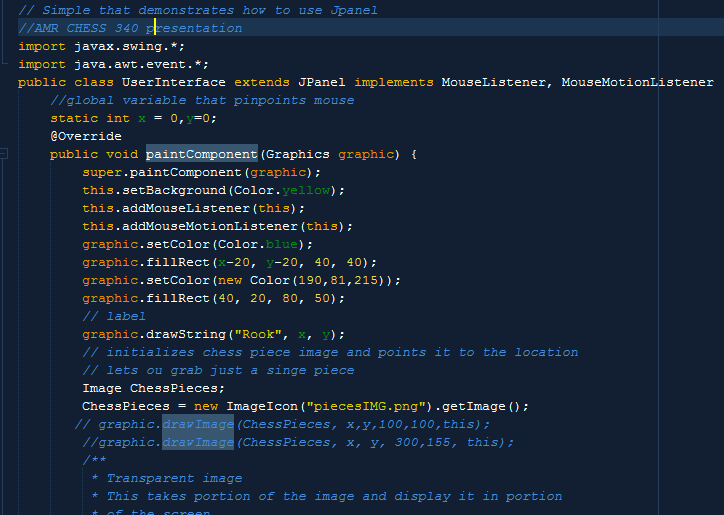
These JFrame and JPanel classes are extension points for Java User Interface Design.

Jframe is used to represent things a window should have. This includes the title bar, minimize / maximize, close, and various event handlers like EXIT\_ON\_CLOSE which exits when you click on the windows Close button.

Jpanel is more important when working with User Interfaces. This is a generic class that is used to group components together. This is especially useful when working with layout.

In this particular project for example, Jpanel was used when dealing with coordinates of the image/pieces and how they are handled and displayed. Coordinates are very tiny and lie between pixels of the output visual layout, if we did not use Jpanel correctly, in this instance, drawImage, when dragging the pieces, we would end up with something like this. 

So Jpanel had to be extended in order to coordinate with the mouse event JPanel Sample code



As you can imagine, dealing with the GUI independent of the logic is not as complicated. The challenge was introduced when the View and the Model had to be joined and make them interact with each other.

For example it is not part of the GUI to determine the game state and which pieces can be moved by the user, but it is part of the GUI to represent those pieces and respond when they are trying to move.

To overcome this challenge we had to create two classes that will merge the logic and GUI together. Since the pieces and board is represented by x and y coordinates, the model part of the program had to take those coordinates and make sense of them.

Sample code for conversion.

