Code Review – penalty checking/action

Architecture and Design
From examining the code, it appears that all of the logic for this is handled upon the culmination of the turn, i.e. in the *reverseTurn()* method. The view will only enable the Continue button after it checks if there has been a penalty. If there is no penalty, it continues to update the game board via *updateGUI()*, and has the controller pass through all interactions with *togglePeg()*. Otherwise, gameplay is paused until *endWhiteJump()* is called. From the perspective of the graphical view, there is little logic to do, other than pass through clicks to pegs made in the GUI, and process the Continue button.

Structure, Code, and Comments
There is a little confusion on the structural flow of how the penalty is exacted, as code control jumps about the place in order to determine the case for a penalty. The use of polymorphism to break apart the player move processing between base Model class and SaveTheNetworkModel class also impedes readability. There are very few comments in the classes responsible for both the model and the view/controller. This, coupled with the presence of ambiguous method names, makes the code rather unreadable, especially when trying to puzzle out the exact logic of what occurs in the program. For example, *endWhiteJump()* is responsible for handling both the ending of White's turn, in addition to processing the continuation of the game after handling the penalty case.

However, at the same time, there is no logic handling in the in the view nor in the controller part of the application, therefore any other GUI can simply "plug-and-play" into interacting with the model. As one of the aims of the developers working on this project is to follow the Model-View-Controller paradigm, this aspect of the application is very well done.

Feature Correctness
Penalties occur in the following cases:

- If only one white piece has the opportunity to jump, but does not move (i.e. the other white piece moves instead), then that white piece is penalized and removed
- If only one white piece has the opportunity to jump, but makes a slide move instead of a jump move, then that whie piece is penalized and removed

Penalties will not occur in the following cases:

- If both white pieces have the opportunity to jump, provided one of them will make a jump move, there is no penalization
- If only white piece has the opportunity to jump, and it jumps, it will not be penalized
- If neither white piece has the opportunity to jump, then there is no penalization.

It appears that this is what is needed by the client, therefore the implementation of this aspect of the application is feature-complete.