UI and Domain Layer Separation Report

The skunk application has user system interaction. The human player of the game interacts with the user interface (UI). This user interface is referred to as the presentation layer (PL). The game execution is done in another layer referred to as the domain layer. The game of skunk has many states based on the dice and the players responses. I found it very difficult to separate the PL and DL layers. The goal is to separate (or decouple) these two layers.

There are many advantages to separating the two layers. If the DL is decoupled from the PL, it is much easier to change the PL without needing to modify the DL. For example, perhaps the PL is moved from command line to a graphical user interface (GUI). Ideally, the GUI can use the same interface between to the DL. Another example would be, perhaps the PL is mapped to Spanish. Again, the hope would be that the DL does not need to be modified.

The opposite is true, perhaps a new or better algorithm is found for calculating the score. The DL should be able to be modified without having to change the PL. This make the game much easier to maintain (improve and fix bugs) and test.

Another advantage of separation was testing. It was much easier to test high level DL objects since there was no input and output of strings.

In project one, I found it very difficult to separate the PL and DL. I have always just created a program and made it do what it needed to do without any consideration to decoupling the PL and DL. In the first implementation of Skunk, the PL had a lot of game control in it. This game control belongs in the DL. In my case, this control logic really belonged in the controller.

I asked to redo the project one as my second project so that I could do a better job with TDD, coverage and separation of the PL and DL. As it turned out, this was a good decision because I was able to see what I had done wrong and fix. If had I created a new project, I would have likely gone down some of the same paths.

In the second implementation, my first attempt at separating the PL and DL was done wrong. I had moved the control out of the PL and put it down in the controller. That was correct. However, I came up with the great idea that I would pass messages from the control logic to the PL. I was thinking “well a string is an object”. I will simply pass that object to the PL. But Professor Level pointed out that I just move the PL to the DL. Passing strings for control is a bad idea. For example, what if a Spanish interface was needed. I would have to change the UI and the controller. The controller is clearly part of the DL. So, on to my third attempt.

In discussions with Professor Level, the light finally came on. I needed to pass objects between the PL and DL that would not need to be modified if ether layer changed. In the third attempt, I created a message board and scoreboard. The message board contained all the string messages. The message board became part of the PL. This isolated the messages so that if the UI needed to change, the change would only need to be made in one place in the code. This worked well. The controller has a state. That state could be passed to the PL and used to look up the correct message. This was a perfect use of a HashMap. The state was the key to the message (string). The PL did not have to understand the state control. It simple displays a message based on what ever state is pass to it. That was nicely decoupled.

The PL need to display more than just state based messages. It needed to display players names, scores and chips based one information that the DL. Professor Level had discussed using some sort of scoreboard to do this. Again, the light finally came on. That is what I needed to do. I could pass the scoreboard from the game to the PL. The PL could extract the information like the players name, game points, chips and what they last rolled without needing to understand anything about the game’s calculations and execution. If the PL switched to a GUI, this same scoreboard could be used to extract displayed fields. Nicely decoupled again.

I do have one thing left to decouple. The PL sends the responses from the user to the controller. The controller executes based on these responses. Currently, the PL passes strings to the controller. It can pass “yes”, “no”, or a player’s name. This should be changed to an object. The object could have two data fields: one of type boolean for a yes or no response and one of type string for a player’s name. This would completely decouple the PL and DL.

Making a simple change to the application would require tests to be modified. Late changes to code have bigger impacts than early changes. I plan to make these changes before the end of the semester.

In conclusion, separating the PL and DL is not easy to do. I think the more you practice it the better at it you become. It is clearly worth taking the time to do. If you write a program, it will need to be maintained. It may be reused. The UI will likely change over time. Separating PL and DL will save time in the long run. I personally will strive to always do it.