Code Swap Review
Group 100
Joe, Simon, Cameren, David

Upon looking at the candidate code made available to us there were a few specific things we were looking for when making our decision. Primarily, we wanted to compare functionality and ease of use of the program itself to readability and overall "cleanness" of the code.

One of the candidates, Candidate A, was very easy for us to eliminate from our decision, as that was the code we had designed for Stage 2 of the project. Although this code was rated most highly by us in overall design, readability, and ease of use, we decided to forgo a failing grade for Stage 3, and select a different candidate.

When looking at Candidate B, we were very impressed with the detail in the code. It was very well documented, and the program itself ran fine. Actually, it had some of the stage 3 functionality (formula cards) already in play, which would have made things very easy for us. What turned us off, however, was that the code itself was much different than what we had designed originally and therefore seemed far more complicated. We decided that if we were to make changes to this code, it would be much more involved than some of the other codebases we had to work off of, and thus decided to not select Candidate B.

Candidate C was knocked off the list rather quickly for us. Upon opening the program and seeing that each tile was composed of 9 JButtons, resulting in an absolutely massive board with plenty of room for errors, it was an easy decision to go ahead with a different codebase. Upon looking at the code, it seemed to be well documented and fairly easy to read through, however altering the code would have involved drastically altering the GUI and the way the Tiles worked in order to look and play like something we wanted it to. For these reasons, we decided against Candidate C.

Candidate D resulted in a similar reaction for us to Candidate C. Opening the program caused a list of runtime errors right off the bat, in addition the program itself did not seem to function. We could

not figure out a way to move pawns or insert tiles, and there was no extra tile for insertion or rotation. So without even looking at the code we could tell there would be a good amount of extra work in getting this Candidate code to function. Upon looking at the code we found it was rather well documented and organized. However this was not enough to justify the additional work needed in adding functionality that should have been in place from Stage 2 and was in place already for other Candidate code. For these reasons, we decided against Candidate D.

Candidate E was the Candidate we decided to move forward with. To begin, running the program worked. Extra tiles were able to be inserted and rotated, pawns were able to move in the proper order, there were instructions as to who's move it was and if they were able to move legally, and the overall design of the program was easy and intuitive to use. Upon looking at the code we were pleased to find it was well documented, the methods were named in a way that made it easy to add functionality or manipulate existing functionality. We also noticed that the code itself seemed to be rather similar to our implementation, which would make manipulating and reading it even more convenient. For these reasons we decided to move forward with Candidate E for our Stage 3 implementation.