EECS 397 Assignment 7: Codebase Swap Keith Kravis and Jayden Soni

We have decided to move with with the team1-working codebase. After going through both codebases, we determined that our implementations were extremely similar: we both used Java and the IntelliJ IDE, and also used similar logic to represent player locations on the board. For the differences we encountered, we think that either the team1 implementation is preferable, or that it will not be very difficult to add to the team1 codebase.

There were many similarities between the approaches that were taken in both the code bases. Board positions were similarly represented with a row integer, a column integer, and a third integer representing the location on a specific tile. Although team1 used an additional abstraction with the BoardSpace class, this representation was consistent. The board representations were also similar in the invariant that players/tokens will always rest on a space that has no tile or they will be eliminated. This allowed for both implementations to avoid the use of phantom tiles and led to both code bases having similarly structured place_tile loops for moving players forward on the board. Tiles were also similarly represented with the use of four pairs of integers to characterize the paths that are on a particular tile (although team7 uses a more direct representation with a 2D array than team1). Finally, much of the structure in the Server (team7) and Game (team1) classes had similar implementations and approaches particularly with regards to the ordering of processing events during a turn.

In general, we found the team1 codebase to be better organized and more readable that the team7 base. The commented sections for instance variables, public methods, etc. make the codebase easy to digest and add to. Team1's BoardSpace and TileConnection classes provide a helpful level of specification for player locations and tile patterns beyond the generic arrays in team7's codebase. It is currently unclear how the difference in DragonTile implementations will play out in future assignments, but thus far, team1's implementation is definitely cleaner: it simply tracks a dragonPlayer instead of maintaining an actual dragonTile object to denote the player. This simplifies the semantics of dealing with playable tiles in player hands. Team7's Turn class is a good way to keep track of the important variables associated with a game, though it may be obsolete within team1's singleton Game class. It could easily be added in if deemed helpful. Finally, team1's Token class serves a similar purpose to team7's SPlayer class, but is better tied in to the way the Board class keeps track of player locations. One drawback of Team1's codebase is the way APlayer's are both responsible for both interfacing with computer, human, and network players and keeping track of player's hands. Some of this functionality may need to be transferred to the Token class in future assignments to better line up with the interface specifications from assignment 5.

A major benefit of the similar approaches of the two code bases is that design choices in team7 can be easily brought over and implemented in the team1 starting point used for the working code. The abstractions around players and machine player implementations were very different in both code bases and this may be one of the harder areas to merge code bases.

However, in other areas such as the board and server functions, it should be very manageable to combine the best aspects of both implementations. The similarities also allow us to merge the test suites from both code bases to account for more cases and game situations.