Jenny Lam, Daniel Bang
CS 393 Software Construction
Professor Dimoulas
1st November 2019

Team47 commits to this code base!

Both of our previous groups have implemented the assignments in Python code, so there is much overlap in terms of knowledge of the language and libraries used in each code base. For the most part, both code bases import sys and json to handle reading and writing json for input and output. The codebase that we chose (team 5) also imports enum to support creating enumerations for the stone type to enforce that stones can only be of type black, white, or none. This helps with interface control, and also makes the codebase more flexible to changes in the representation of stones whereas the other code base (team 11) uses strings “B”, “W”, and empty string throughout the code without this level of abstraction.

Based on the results of the latest Travis run on peer tests, the codebase that we chose was able to pass all of the peer tests for 5.1 and 5.2, whereas the other codebase currently does not run to completion for 5.2 due to its less optimal implementation for the player that prioritizes guaranteed capture within its strategy’s n moves. Team 5’s implementation of the second player keeps track of groups of connected components on the board. This allows them to only have to check the liberties for a group once, but team 11 iterates through all of the opponent’s stones that occupy the board and check the liberties for each one using a breadth first search. This second method requires duplicate work because even though a point is in a connected set, the iteration means it will go through every opponent stone regardless of previous checks. Additionally, team 5’s codebase includes code that formats json output and makes it more readable for debugging purposes. Going forwards, this feature and the level of abstraction team 5 was able to achieve will be easier to build upon.

In terms of design, team 5’s codebase has a distinct advantage over team11’s codebase because of its modularity. In addition to having constants, the codebase separates each object class into its own file for better organization. These classes divide public and private functions and does a very good job of mapping one purpose to each of the functions for increased readability and ease of debugging. In particular, for validate history in the rule checker, team 5 separates out the functionality into checking for valid board history and checking for move validity based on its history, whereas team 11’s validation puts everything into one function. Team 5’s organization makes code extremely reusable, meaning we can import it using import features of Python every time we need to access object classes created for a previous assignment. Overall, team 5’s codebase design is more mature and follows a lot of good stylistic conventions, including modularity, and thus provides a good base for upcoming assignments.