Team38 commits to using the Team21 codebase!

1. Both group members have been using Python for the first part of the quarter. None of the libraries that are used in either of the codebases are unknown to us. Additionally we saw that both structures are rather similar making it easier for both team members to transition to the new codebase.
2. Team 21's codebase features a more succinct way of checking for valid board history than Team 19's code. Team 19 uses a merge_board function that checks for valid removes in the histories. Team 21 does this without having to rewrite a whole board and just comparing changed points. Team 19 breaks down a lot of bigger functions into smaller helper functions, a feature we intend to better implement into Team 21's codebase. Both codebases lack unit tests, so we will be spending some time to include those to ensure any refactors we do maintain the correctness of the code. Team 21's code also includes an extensive global variables file, that will be useful when changing the size of the board. The Player class in Team 21's codebase contains the methods for making moves. This is an organizational choice that makes the most sense and we all will be checking the board history from this class.
3. Team 19 Design:

    Json Parser parses the input from STDIN and adds each parsed python element to a list that it then sends to the Go class. The Go class handles the python objects and executes the appropriate methods inside of the Player class. Inside of the Go class, rule-checking is performed by instantiating the HistoryBoards class which verifies that the board's being passed obey the rules of Go. From the Player class any moves are performed and the appropriate response is sent back to the frontend where it is output in json format to STDOUT. The methods and data representation of the Board and Point class are stored internally and cannot be accessed from outside these respective classes.

    Team 21 Design:

    Frontend parses the input from STDIN and adds each parsed python element to a list that it then sends to the Game class. The Game class handles the python objects and executes the appropriate methods inside of the Player class. The Player class performs rule-checking by calling individual methods within the rulechecker file. From the Player class any moves are performed and the appropriate response is sent back to the frontend where it is output in json format to STDOUT. The methods and data representation of the Board and Point class are stored internally and cannot be accessed from outside these respective classes.