Team45 commits to this code base!

We decided to commit to team23's code base.

(i) This code base is written in Python, using libraries such as json, sys, queue, and copy. Both Ben and I are familiar with Python and the libraries used in this code base. Also, Ben's knowledge of contracts library in Python can be a great addition to the existing code base. Since other code base is also written in Python with similar libraries to this code base, our selection was made based on the organization, readability, and correctness and less on the language and libraries the code base was implemented in.

(ii) This code base implemented all the functions that belong to the public interfaces of previous assignments. We are quite sure that all the functions work the way it should work because of the unit tests written for the functions. Moreover, the reason we chose this code base is that it's more readable in a sense that each function in this code base serves a single purpose and minimizes side-effects. Also, a unique feature that this code base provides is that this code base has a function that gets all the chains and associated liberties given a board. This function can be extremely useful for two reasons. First, it avoids repeating computations. Second, it gives useful insights into the board, which can be helpful when computing game score and selecting a move.

(iii) As for the design of this code base, we have four files that represent each component of the Go – point_utils.py, board_utils.py, rule_checker.py, and player.py – along with some other utilities files such as parser.py. Each file is named carefully to fit the purpose of each component of the Go. Moreover, all the functions in the file are appropriate to the purpose of the component, and we do not have functions that should belong to some other components. One design strength of this code base is we have separate strategies.py file that contain different strategies for the move, which can be given to the player at the initialization of player object. This separation of player and strategy can be useful when we have many strategies and having them inside the player.py would be a less modular design choice. Also, all the files import constants from constants.py file. The constants are basically the settings of the game such as BOARD_ROW_LENGTH, BOARD_COL_LENGTH, STONES, MAYBESTONES, B, W, etc. This makes our code very flexible to changes in the game parameters. For example, if we want to test our functions on the smaller board, we can simply change board related constants in constants.py file, without having to go to all the files to modify the board sizes one by one.