1 Assignment

Summary: Write C++ code to determine the state of a tic-tac-toe board and write tests that validate the correctness of your code. We will provide the function interface for your code, but the design of the implementation is up to you. Try to design and write your code to be as clean and readable as possible, using what you read in Chapters 1, 2, and 3 of the book to inform your implementation.

Background: Tic-tac-toe (`https://en.wikipedia.org/wiki/Tic-tac-toe`) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3x3 grid with their mark. Player X is the first to make a mark. Once a space has been marked, it cannot be marked again. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game, and the game is over (i.e., no further actions occur).

Specification: The provided function interface expects a description of the state of a tic-tac-toe board in the form of a `std::string`. This string should consist of 9 characters, one for each position of the board. The characters are ordered in groups of three characters (left-to-right) specifying top, middle, and then bottom horizontals, as shown in Figure 1. The string is case-insensitive. Squares marked by player X are specified with an 'x' or 'X' and squares marked by player O are specified with an 'o' or 'O'. Any other character is considered to be an empty square. Figure 1 shows what a board specified as "o-XxxoO.z" would look like.

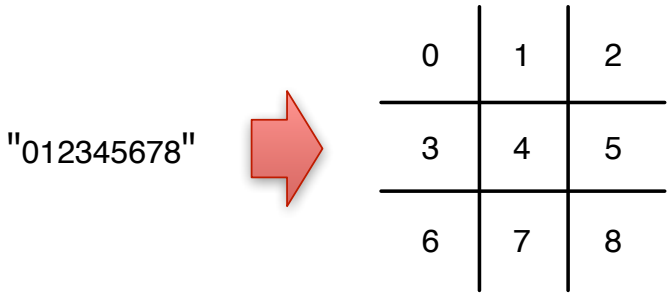

Figure 1. The mapping from Strings to positions in the Tic Tac Toe board.

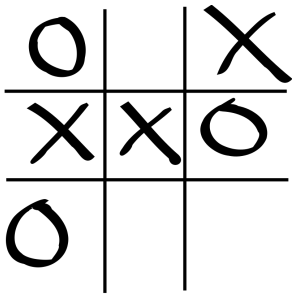

Figure 2. The above Tic Tac Toe board could be described as: o-XxxoO.z

Your implementation of `evaluateBoard` should analyze the state of the specified board and return one of five `Evaluation` values. The values should be defined as a C++ `enum`. If you aren't familiar with `enums` in C++, there are a lot of good references on the Internet; it isn't that difficult of a concept.

If the argument passed to the function doesn't correctly describe a board, your code should return `InvalidInput`. If the argument describes a board state, but one that is not reachable by playing a game of tic-tac-toe, then your code should return `UnreachableState`. If the function's argument describes a valid, reachable board state, then it should return `Xwins`, `Owins`, or `NoWinner` if the board is in a state where X has won, where O has won, or where no one has won, respectively.

Your repository: Use the following link to create your own copy of the TicTacToe repository on GitHub:

`https://classroom.github.com/a/BL0sBZgv`

Testing: The primary motivation for this assignment is to give you a moderately complicated function for which to write black box tests. Following a test-first philosophy, we're going to ask you to write your tests before writing your code. More precisely you should have tests checked in to the repository before you have code.

Your tests should be implemented in the provided `TicTacToeTest` class using Catch2. Each assert should be in a separate test. Name your tests with meaningful names. For most tests, no comments are necessary, because the name of the test is sufficiently explanatory. Write enough tests to give yourself confidence that your implementation will be correct if it passes all of the tests. Exhaustive testing is untenable, as each of the nine positions of the board can be X, O, or blank, leading to 3^9 (over 19,000) board configurations.

To effectively test your code using a much smaller number of tests, use the "Bag of Tricks" we discussed in Lecture 2 (e.g., equivalence classes, boundary conditions, classes of bad data) to pick a useful set of inputs to test. Take an adversarial approach to writing your tests, trying to identify the problematic inputs that are likely to break your implementation and that of other students. We'll be gathering together all of the tests that the class writes and testing everyone's implementations with all of those tests that pass on our reference implementation. We'll award a small amount of extra credit to students whose implementations pass the most tests and write valid tests that break the most other student implementations.

Catch2 can be found at the following URL:

`https://github.com/catchorg/Catch2`

Design and Style: As you write your code, use the best design and coding style that you are familiar with. In particular, for this assignment, try to use good names (as directed by Chapters 2 and 3 of the text book) and try to avoid replicating regions of code ("Don't repeat yourself!").

Also, we'll be using the Google C++ Style Guide for the code that we write in CS 126. For this assignment, please read the portion of the style guide associated with naming (see the URL below) and follow the conventions described.

`https://google.github.io/styleguide/cppguide.html`