Lab 2:
Queen's Gambit

Due February 13 by the start of lecture.

Overview

In this lab, you will write test cases for my chess implementation following proper TDD practices. Your tests will be shared with other students to aid in the development of their own chess implementation in Project 1, except for a single "buster" test that you will design to trip up as many of your peers as possible.

Getting Started

Download the file `lab2_files.zip` from BeachBoard. Extract the files to your computer and either open the `.sln` file in Visual Studio 2017 or the root folder in Visual Studio Code.

Writing Tests

Using the file `ExampleTests.cs` as a reference point, you must write **six test cases** for the `ChessBoard` class:

- At most 1 test can be about the initial starting board state. (We don't need 100 test cases that count how many pawns are on the starting board.)
- At least 1 test must involve a "tricky" move, like castling, pawn promotion, or en passant. (It's easy to mess these up.)
- At least 2 tests must involve `UndoLastMove`. (Check many aspects of the board after the move is undone.)
- One test **must** involve validating `GetPossibleMoves` to see that a particular piece has correctly reported **all** its possible moves. Design this test to stress the piece movement logic – with either very few or very many possible moves. Don't pick a piece at random; instead, **take your student ID, modulus it by 5, and add 1:**
  - for 1, test rooks
  - for 2, test knights
  - for 3, test bishops
  - for 4, test queens
  - for 5, test kings
- At least 1 test must place a king in check, and validate the possible moves that result. (This is a difficult thing to get right.)
- Half of your tests should involve the white player, and half should involve the black player. (They have related but opposite logic for some pieces, like pawns.)
- **All of your assertions must contain string explanations**, and all your test case methods should contain a comment that describes the general purpose of the test.

Validating your tests:

You must validate all your assertions by hand; there's no point in including a test case whose conclusion is actually incorrect. All the tests you write must **pass** when run on my chess code, unless...

If you think my chess code is incorrect:

If you write a test that you **swear** enforces the correct behavior of a chess game, **but** my chess code fails on the test, then please email me ASAP... but double check your work first. I'll give prizes to anyone who finds a bug in my code and presents it to me.

Buster test:

In addition to your six test cases above, you must write one **buster test** that will **not** be shared with your peers. Your buster case should test something especially complicated, or that you think your peers might overlook. When Project 1 is assigned, students will submit their work to a grading robot that executes the buster test cases. Whichever buster test case fails the most student submissions will earn a prize at the end of the semester. **Please name your buster method according to the rules outlined in the file `MyChessTests.cs`.**

Deliverables

By the due date, you must upload your `.cs` file with your name to BeachBoard's DropBox folder for Lab 2. That's all you have to do. Do not give me any of the other files.