CPSC 3720 – Mocking
(Pawn Game)

Overview

In this assignment, you will:

- Demonstrate your knowledge of using mocking for testing by testing a "Controller" in an application that uses the Model-View-Controller architecture.
- Creating mocks for the dependent clases (i.e. a Model and a View).
- Keep track of your progress using version control.
- Write passing unit tests for the class that test the behaviour of the Controller.
- Determine how well your code is tested using code coverage.
- Maintain a coding style with a style checker.
- Check for memory leaks using a memory checker.
- Use static analysis to detect bugs and avoid dangerous coding practices.
- Use continuous integration to automate the running of software engineering tools.

Overview of the Application

The Application Under Test (AUT) is a simplified version of the game of chess. The game is played on a 5x5 board, as opposed to an 8x8 board, and only pawns are used. Each player's pawns start on the back rows (i.e. rows 1 and 5). The rules that govern a pawn's movement remain relatively the same as in chess:

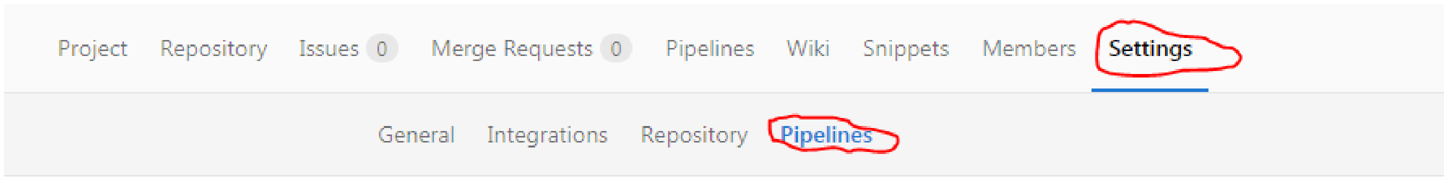
- Pawns can only move forward one space, except on their first move when they can move two spaces.
- A pawn that moves two spaces on it's first move to "avoid capture" by an opponent's pawn can be taken "en passant" (see https://www.chess.com/terms/en-passant).
- There is no promotion of pawns when the reach the opponent's back rank, as there are no other pieces for which the pawn can be promoted.
- The game is over when all of the opponent's pawns can be captured (i.e. "last man standing").
    - For simplicity of the assignment, we'll assume that players will not move in a such a way as to cause a stalemate where neither player will be able capture the opponent's remaining pawns.

Instructions

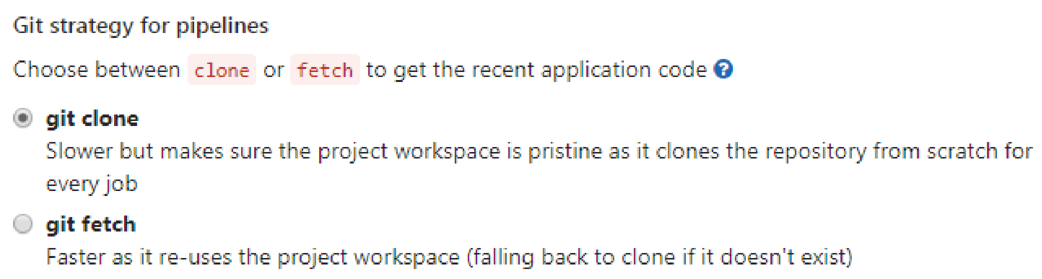
Setup

1. Fork the assignment repository so that you have your own GitLab repository for completing the assignment.
    a. If you do not do this step, the marker will not find your assignment repository, and **you will receive an automatic 0 for the assignment.**

2. Set up your GitLab repository for running continuous integration on your project.



a. Set the *Git Strategy* to "`git clone`"



Git strategy for pipelines

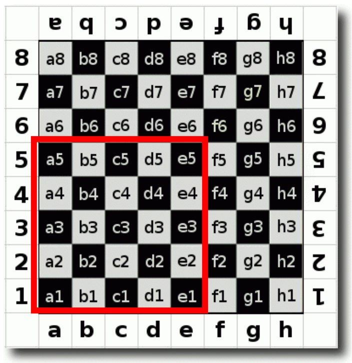
Choose between `clone` or `fetch` to get the recent application code

● **git clone**
Slower but makes sure the project workspace is pristine as it clones the repository from scratch for every job

○ **git fetch**
Faster as it re-uses the project workspace (falling back to clone if it doesn't exist)

Completing the Assignment

1. Create a local clone of your assignment repository.
   a. Run the command `git remote` and verify that there is a remote called `origin`.
      i. `origin` is the link to your repository of GitLab and is where you will be pushing your changes.
2. Create a unit test that uses mocking to test the behaviour of the `playGame()` method of `Chess`.
   a. You will need to create mock classes for the Model (i.e. `Board`) and View (i.e. `ChessUI`).
3. The behaviour of the `playGame()` method proceeds as follows:
   a. The method `setup()` is called to place the pieces on the board (i.e. `Board.placePiece()` is called for each piece).
   b. The board is drawn by calling `Board.draw()`. Also, this occurs after a player moves.
   c. The players will alternate moving their pieces starting with the player controlling the white pawns. `ChessUI.getLocation()` is used to get the location of the piece to move and where to move it. Players enter board coordinates using algebraic notiation (i.e. the lower left square is "A1" and the upper right square is "E5").



   d. A piece is moved by calling `Board.movePiece()`.

e. The game checks to see if there is a winner by calling `Board.checkWinner()`.

f. If there is a winner, `ChessUI.gameOver()` is called to announce the winner.

4. A `Makefile` is provided to help you build and test your program, run static analysis, memory leak checking, style checking and code coverage.
    a. The Makefile has the following targets:
        i. `tests`: Builds and runs the unit tests.
        ii. `chess`: Create an executable for the game.
        iii. `memcheck`: Runs `valgrind –memcheck` to check for memory leaks.
        iv. `style`: Runs `CPPLINT` to check for coding style violations.
        v. `static`: Runs `cppcheck` to check for bugs and bad programming practices.

Example

The following is an example set of moves and actions that can be used for testing the game.

1. White moves from A1 to A3.
2. Black moves from B5 to B4.
3. White moves from A3 to B4.
4. `Board.checkWinner()` returns `true`;

Grading

Based on your demonstrated understanding of unit testing using mocking, version control, and good software engineering practices, you will be graded. Examples of items the grader will be looking for include (but are not limited to):

- Unit test(s) that use mocking to test the behaviour of `Chess.playGame()`.
- Proper use of version control.
    - Version control history shows an iterative progression in completing the assignment.
    - Version control repository contains no files that are generated by tools (e.g. object files, binary files, documentation files)
- The status of the most recent build in your repository's GitLab pipeline nearest the deadline (but not after the deadline) is green (i.e. passes).
    - Memory leak checking and style analysis show no problems with your code.
- Testing code contains no "dead code" (i.e. code that is commented out).

Submission

There is no need to submit anything, as GitLab tracks links to forks of the assignment repository.

- Ensure that the permissions are correctly set for your repository on GitLab so the grader has access. **You will receive an automatic 0 (zero) for the assignment if the grader cannot access your repository.**