CPSC 3720 – Assignment 1

Overview

In this assignment, you will:

- Design and implement a text-based card game using Model-View-Controller.
- Keep track of your progress using version control.
- Write passing unit tests for Model classes.
- Write passing unit tests using mocking for the Controller class.
- 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.
- Generate documentation for your code using doxygen.
- Use continuous integration to automate the running of software engineering tools.

Instructions

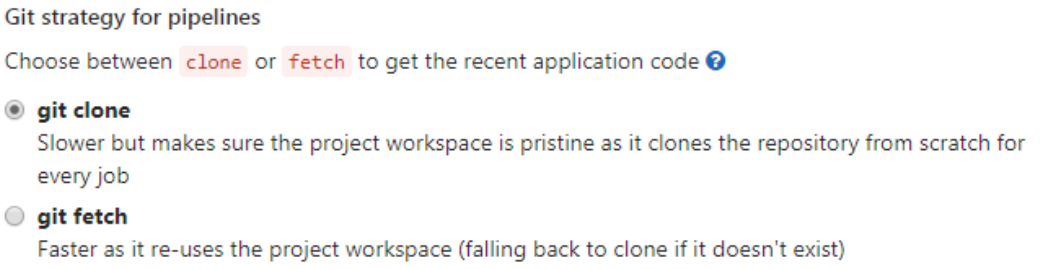
Setup

1. Fork the repository so you have your own GitLab repository for completing the assignment.

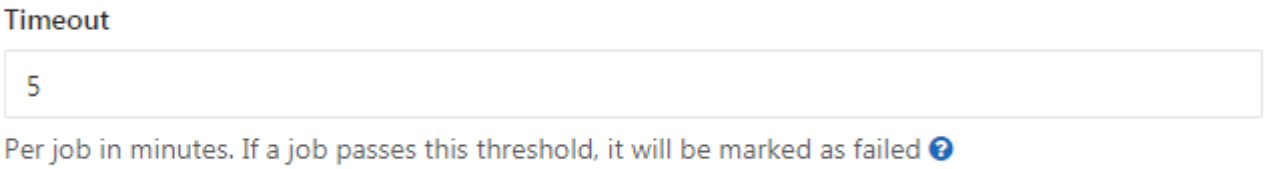
    a. If you do not do this step, the marker will not be able to find your assignment repository and **you will receive an automatic 0 for the assignment.**

2. Setup your GitLab repository for running continuous integration for your project.

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



b. Set the Timeout to 5 (i.e. 5 minutes). Your CI job will be small, so this should be lots of time and will prevent any infinite loops from tying up the CI server.



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 text-based game of Go Fish. (If you are not familiar with the game, see https://bicyclecards.com/how-to-play/go-fish/)
    a. The computer will handle dealing.
    b. The design of your system must use the Model-View-Control pattern.
    c. Create unit tests using Google Test (`gtest`) that test your *Model* classes.
        i. Your model classes are to have as close to 100% statement coverage as you can get.
    d. Create unit tests using Google Test & Mock (`gtest` & `gmock`) that tests your *Controller* class.
        i. Your controller class is to have as close to 100% coverage as you can get.
    e. As the *View* class handles interaction with the user (i.e. `cin`), do not test it using automated testing – it will hang the CI server.
    f. An example `Makefile` is provided to help you build and test your program, run static analysis, memory leak checking, style checking and code coverage. Edit the `Makefile` so that it works for your assignment.
        i. The Makefile must have the following targets:
            1. `cardGame`: Builds the game for system-level testing.
            2. `testGame`: Builds and runs the unit tests.
            3. `memcheck-game`: Runs `valgrind --memcheck` to check for memory leaks when playing the game.
            4. `memcheck-test`: Runs `valgrind --memcheck` to check for memory leaks when running the unit tests.
            5. `coverage`: Runs `lcov` to generate HTML reports of the unit testing code coverage.
                a. The HTML reports are to be located in the `coverage` directory.
            6. `style`: Runs `CPPLINT` to check for coding style violations.
            7. `static`: Runs `cppcheck` to check for bugs and bad programming practices.

8. `docs`: Generates HTML documentation using `doxygen` for your application.
    a. Use `doxywizard` to create your `doxygen` configuration file. Make sure any paths are relative, not absolute, otherwise the marker will not be able to generate the files after cloning your repository.
    b. Generate the HTML into the `docs` directory.

Grading

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

- The design of your system. It should be clear that the `Model-View-Controller` pattern was used.
    - Consider naming your files `GoFish` (controller) and `GoFishUI` (view).
- Public methods of `Model` and `Controller` classes are tested by unit tests and mocking.
    - Unit tests show the use of equivalence partitioning in the creation of test cases.
    - Statement coverage is as close to 100% as can reasonably expecting from looking at the `lcov` report.
- 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 most recent build in your repository's GitLab pipeline nearest the deadline (but not after the deadline).
    - Memory leak checking, static analysis and style analysis show no problems with your code.
- Playing your game. The prompts for your game should be clear and concise.
- Generated documentation shows all public classes, enums, constants, and methods are documented.
- Source 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.

- Make sure 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.**