Old Maid (TEXT-BASED)

Due 1st December 2024

Overview:
In this assignment, you will:

- Design and implement a text-based card game.
- Keep track of your progress using version control.
- Write passing unit tests and 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

This is an individual project, NOT a group project

Fork the repository (project2) on gitlab so you have your own copy.

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.**

Set the project visibility for your forked repository to "Private".

a. This means that **no one else** will not access to your work unless you give them permissions

Add the marker, lab grader (Deen), and Dr. Abdoulahi (ibrahim) as a member of your repository with the permission "**Reporter**".

Completing the Assignment

Create a text-based game of Old Maid.

a. If you are not familiar with the game, see https://bicyclecards.com/how-to-play/old-maid/.
b. To make things easier for you, I am providing the interface of the game (see the include folder)
c. Create unit tests using Google Test (gtest) that test your game.
d. Have as close to 100% code coverage as you can get.

The Makefile must have the following targets:

a. cardGame: Builds the game for system level testing.
b. testGame: Builds and runs the unit tests.
c. memcheck-game: Runs valgrind –memcheck to check for memory leaks when playing the game.

d. memcheck-test: Runs valgrind –memcheck to check for memory leaks when running the unit tests.
e. coverage: Runs lcov to generate HTML reports of the unit testing code coverage. The HTML reports are to be located in the coverage directory.
f. style: Runs CPPLINT to check for coding style violations.
g. static: Runs cppcheck to check for bugs and bad programming practices.
h. 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, bug reporting, and good software engineering practices. Examples of items the grader will be looking for include (but are not limited to):

- Public methods of your classes are tested by unit tests.
  - Unit tests show the use of equivalence partitioning in the creation of test cases.
  - Coverage is at least 80%.
- 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.**

PLATFORM

The project must run in the Linux environment of the University of Lethbridge cs.uleth.ca network.

NOTE

No files that can be generated when compiling and running appear in the repository, including executable files, code coverage reports, the documentation generated by Doxygen, or files created by VSCode/Git Bash/other software.