Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Métodos de Programação
Código da Disciplina: 201600

Métodos de Programação - 201600

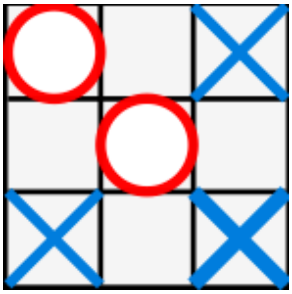
Trabalho 1

O objetivo deste trabalho é utilizar o desenvolvimento orientado a testes (TDD) para resolver o problema de verificar o jogo da velha. As regras do jogo da velha são dadas em: http://portaldoprofessor.mec.gov.br/fichaTecnicaAula.html?aula=28141

O jogo da velha é representado como uma matriz 3x3 de inteiros.
O valor 0 significa que a posição está vazia
O valor 1 significa que a posição está com um X
O valor 2 significa que a posição está com um O

Ex. A matriz
[2, 0, 1]
[0, 2, 0]
[1, 0, 1]

Representa o jogo:



O objetivo é fazer e testar uma função que:

Tem como parâmetro a matriz 3x3 inteiros
Retorna 1 se o vencedor foi o X
Retorna 2 se o vencedor foi o O
Retorna 0 se o jogo está empatado
Retorna -1 se o jogo está indefinido (ex. tem apenas um X)
Retorna -2 se o jogo é com certeza impossível pelas regras (ex. todas as posições são X)

O desenvolvimento deverá ser feito passo a passo seguindo a metodologia TDD. A cada passo deve-se pensar qual é o objetivo do teste e o significado de passar ou não no teste.

1) O programa deverá ser dividido em módulos e desenvolvido em C ou C++. Deverá haver um arquivo velha.c (ou .cpp) e um arquivo velha.h (ou .hpp). Deverá haver também um arquivo testa_velha.c (ou .cpp) cujo objetivo é testar o funcionamento da biblioteca de verificação do jogo da velha.

Faça um Makefile utilizando os exemplos já vistos.

O programa e o módulo devem ser depurados utilizando o GDB.
(http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html)
(https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf)

2) Utilize o padrão de codificação dado em: https://google.github.io/styleguide/cppguide.html quando ele não entrar em conflito com esta especificação. O código dever ser claro e bem comentado. O código deve ser verificado se está de acordo com o estilo usando o cpplint (https://github.com/cpplint/cpplint).

Utilize o cpplint desde o início da codificação pois é mais fácil adaptar o código no início.

3) Faça um documento (.txt ou .pdf) dizendo quais testes você fez a cada passo e o que passar neste teste significa.

4) O desenvolvimento deverá ser feito utilizando um destes frameworks de teste:

4.1) gtest (https://code.google.com/p/googletest/)
4.2) catch (https://github.com/philsquared/Catch/blob/master/docs/tutorial.md)

5) Faça a análise estática do programa utilizando o cppcheck, corrigindo os erros apontados pela ferramenta.

Utilize cppcheck --enable=warning .
para verificar os avisos nos arquivos no diretório corrente (.)

Utilize o cppcheck sempre e desde o início da codificação pois é mais fácil eliminar os problemas logo quando eles aparecem. Devem ser corrigidos apenas problemas no código feito e não em bibliotecas utilizadas (ex. gtest, catch)

6) Deve ser utilizado um verificador de cobertura

ex. gcov. (http://gcc.gnu.org/onlinedocs/gcc/Gcov.html). O makefile deve ser modificado de forma incluir as flags -ftest-coverage -fprofile-arcs. Depois de rodar o executável rode gcov nomearquivo e deverá ser gerado um arquivo .gcov com anotação.

O verificador de cobertura é utilizado para saber qual percentual do código é coberto pelos testes. Neste caso os testes devem cobrir pelo menos 80% do código por módulo.

7) Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc.
Para gerar uma documentação mais adequada, rodar
doxygen –g
isto irá gerar um arquivo Doxyfile. Neste arquivo, na linha adequada, colocar:

EXCLUDE = catch.hpp

Isto fará com que o catch.hpp não seja documentado. Uma mudança semelhante deverá ser feita para outro framework se necessário.

Depois de feito isto para documentar basta roda : doxygen

8) Utilizar o Valgrind (*valgrind.org/*) para detectar problemas relacionados a memória.
Use valgrind --leak-check=full ./nome_executavel

9) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do git (https://git-scm.com/docs/gittutorial)

git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
git init
git add *
git commit –m “teste 1”
git log

Compactar o diretório “.git” ou equivalente em um “.zip” enviar junto.

Você pode inclusive adicionar as verificações dentro do makefile.

Devem ser enviados para a tarefa no aprender3.unb.br um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula_primeiro_nome.zip. ex: 06_12345_Jose.zip. Deve ser enviado um arquivo dizendo como o programa deve ser compilado e rodado.

Deve ser enviado o diretório “.git” compactado como um “.zip”

Data de entrega:

12/ 9 /21

Pela tarefa na página da disciplina no aprender3.unb.br