

Universidade de Brasília  
Departamento de Ciência da Computação  
Disciplina: Técnicas de Programação 2  
Código da Disciplina: CIC0198

## **Técnicas de Programação 2 – CIC0198**

### **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:

[https://sme.goiania.go.gov.br/conexaoescola/ensino\\_fundamental/jogo-da-velha-3/](https://sme.goiania.go.gov.br/conexaoescola/ensino_fundamental/jogo-da-velha-3/)

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

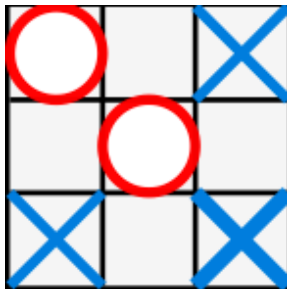
Ex. A matriz

[ 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.

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

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>)

1) 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 deve 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.**

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

`gtest` (<https://code.google.com/p/googletest/>)

`catch` (<https://github.com/philsquared/Catch/blob/master/docs/tutorial.md>)

3) 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 enviando ele junto.

**Para cada teste que você criar, para cada teste que você passar, para toda refatoração, você deve fazer um “git commit –m “nome do teste” “. Este é o principal fator avaliado nestes trabalho, se o desenvolvimento foi feito orientado a teste.**

4) 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.**

5) Utilize um verificador estático

ex. cppcheck, corrigindo os erros apontados pela ferramenta.

Utilize cppcheck --enable=warning .

para verificar os avisos nos arquivos no diretório corrente (.)

**Utilize o verificador estático 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) utilizar o verificador dinâmico Valgrind ([valgrind.org/](http://valgrind.org/))

**Utilize o verificador dinâmico sempre e desde o início da codificação pois é mais fácil eliminar os problemas logo quando eles aparecem.**

Devem ser enviados para a tarefa no [aprender3.unb.br](http://aprender3.unb.br) um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. O documento deve estar na raiz do diretório. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula\_primeiro\_nome ex: 06\_12345\_Jose.zip. Deve conter também um arquivo leiametext que diga como o programa deve ser compilado.

**Deve ser enviado o diretório “.git” compactado junto com o “.zip”. É ele que comprova que você fez o trabalho orientado a teste. O desenvolvimento orientado a teste será**

**avaliado por ele. Verifique desde o início se tem este diretório e se tem a informação adequada.**

Data de entrega:

**24/ 9/ 25**

**Pela tarefa na página da disciplina no [aprender3.unb.br](http://aprender3.unb.br)**