Universidade de Brasília
Departamento de Ciência da Computação
Métodos de Programação 1/2018

Projeto de Disciplina

Datas de entrega:

Definição dos Grupos: 13/06/18 ate as 23:55

Final : 04/07/18 ate as 23:55

Jogo de xadrez

Descrição:

Implementar um programa que permita jogar xadrez de acordo com a especificação a seguir.

1) Interface:

A interface dever permitir:

a) Começar um jogo novo:

começar com o tabuleiro vazio que pode ser editado em qualquer configuração

começar com um tabuleiro padrão. Pode ser escolher começar com as peças brancas ou com as pretas.

b) Jogar:

Pode jogar contra outro adversário humano. As jogadas são alternadas e só podem ser feitas jogada válidas de acordo com as regras do xadrez.

As jogadas podem ser feitas direto na interface movendo as peças ou se digitando a jogada na notação algébrica em inglês (https://en.wikipedia.org/wiki/Algebraic_notation_(chess)).

Tome cuidado com as notações ambíguas (quando uma mesma notação permite entender dois movimentos diferentes). Deve haver uma forma de voltar as jogadas.

c) modo de analise:

O usuário é livre para escolher qualquer jogada válida. O programa diz qual é a melhor jogada a ser feita (brancas ou pretas) para aquela configuração de tabuleiro. Dá uma lista de ‘n’ jogadas ordenadas da melhor para a pior. Cada jogada tem um número associado indicando a avaliação desta.

d) salvar e recuperar o tabuleiro

Uma configuração de tabuleiro pode ser salva em um arquivo ou recuperada de um arquivo. O formato do arquivo deverá ser o PGN (*Portable Game Notation*)

(https://en.wikipedia.org/wiki/Portable_Game_Notation)

e) Funcionamento do programa:

Deve verificar se uma configuração de tabuleiro é valida ou não (ex. se um tabuleiro tem apenas um rei, ela não é valida)

Deve verificar se um movimento é válido ou não

Deve ser capaz de verificar se houve cheque-mate ou não

Deve verificar se houve empate ou não.

f) Mecanismo do jogo.

Para decidir qual é a melhor jogada, o programa deve montar uma árvore de decisão onde os ramos são todos os movimentos válidos do jogador. A seguir, se determina todos os movimentos válidos para o adversário. Depois, cada movimento do adversário, se monta a subarvore com todos os movimentos do jogador. Depois de montar uma arvore de um certo tamanho, se escolhe a jogada que dá o melhor resultado. A árvore pode ter um tamanho que é exponencial em relação ao número de jogadas possíveis. Para tornar a escolha viável com os recursos de tempo e memória disponíveis, se para a árvore e um determinado nível e se atribui um valor para os nós. Estes valores normalmente levam em consideração as peças, as posições destas, possibilidade de cheque-mate, etc. Normalmente, são feitas otimizações para cortar partes da árvore que não dão bons resultados. O mecanismo de jogo deve escolher tentar escolher uma boa jogada.

No modo de analise, o programa constrói a árvore e vai descobrindo a melhor jogada na medida em que a árvore cresce. O programa deve mostrar a melhor jogada dada encontrada até o momento.

Deve existir uma opção de mostrar a árvore do jogo. Ela pode ser muito grande para ser mostrada inteira e deve ter uma opção para só mostrar as ‘n’ melhores jogadas por nível.

O site (https://chessprogramming.wikispaces.com/) pode ajudar como referencia na implementação.

g) Devem ser geradas historietas sobre o sistema descrito. Estas historietas devem ser mapeadas em casos de uso que vão servir para o *backlog* de produto. Devem ser feitos diagramas de caso de uso e de atividade. (diagramas de classe ou seqüência são opcionais)

h) Devem ser feitos testes de cobertura de instruções e de cobertura de caminhos como feitos no **trabalho 4**.

Requisitos:

1) Devem ser aplicados neste trabalho todos os conceitos vistos nos trabalhos anteriores. **O programa pode ser feito em C ou C++**

a) Modularização (ex. makefile, .h e .c).

b) utilize um padrão de codificação:

ex. https://google.github.io/styleguide/cppguide.html

O código deverá ser devidamente comentado facilitando o entendimento.

c) Testes utilizando um *framework* de teste. Ex. Gtest ou Catch. Como estes testes mostram que o programa segue a especificação. **O desenvolvimento deve ser feito orientado a testes**.

d) Devem ser feitas revisões no código conforme visto em sala de aula

e) Assertivas de entrada e de saída. Estas assertivas devem ser colocadas no código através do comando **assert** ou **ifs**. Comentários no código também devem especificar quais são estas assertivas

f) Para cada uma das funções adicionar comentários indicando qual são as interfaces explícita, implícita com a devida descrição, quais são os requisitos e as hipóteses de cada função. Além disto, as funções devem ter finalização adequada liberando memória, fechando arquivos, etc.

g) Faça a modelagem física das estruturas de dados. Use cabeças de estrutura de dados.

h) Assertivas de entrada e de saída como parte da especificação (comentários antes das funções).

i) Assertivas como comentários de argumentação.

j) Assertivas estruturais: elas definem a validade de uma coletânea de dados, ou estruturas de dados, e dos estados associados a estes dados.

k) Coloque nos comentários antes das funções quais são as assertivas do **contrato na especificação.** Diga o que deve ser esperado da função cliente em relação à entrada e o que deve ser garantido pela função servidora na saída.

2) Instrumente o código utilizando um verificador de cobertura ex. gcov.

Neste caso os teste devem cobrir pelo menos 80% do código em cada módulo.

Utilize um analisador estático ex. cppcheck, corrigindo os erros apontados pela ferramenta

3) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do github (https://github.com/)

Um tutorial inicial pode ser encontrado em:

https://guides.github.com/activities/hello-world/

4) O tempo em cada tarefa pode ser facilmente contabilizado utilizando aplicativos ou sites como:

https://toggl.com

5) O projeto pode ser gerenciado utilizando o:

trello.com

6) Interface gráfica poderá ser uma destas:

a) **biblioteca ncurses.**
Ela pode ser instalada no terminal shell do Linux usando o comando

> sudo apt-get install libncurses5-dev

Uma vez instalada, a biblioteca ncurses ela pode ser compilada utilizando o GCC e a diretiva –lncurses.
Dependendo da instalação ncurses pode ser incluída por <ncurses/ncurses.h> ou <ncurses.h>.

Existem vários tutoriais disponíveis como:
http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html

b) **SDL**

https://www.libsdl.org/

Ela pode ser instalada com apt-get e deve ser devidamente configurada e referenciada no código.

c) **PlayCB**

Ela pode ser instalada a partir de:

http://pt-br.playcb.wikia.com/wiki/Wikia_PlayCB

d) **Qt**

Ela pode ser instalada a partir de:

https://www.qt.io/

Se houver interesse em usar outra interface gráfica, o professor deve ser consultado antes.

Módulos

O programa deve ser dividido em módulos.

Observações:

Utilize os princípios de modularidade na criação dos módulos, definidos seus módulos de definição, implementação e organizando suas compilações e ligações (**links**) de forma adequada.

Controle de qualidade das funcionalidades

Deve-se criar um *módulo controlador de teste* (disciplinado) usando o *framework* de teste para testar se as principais funcionalidades e restrições dos módulos.

O desenvolvimento deverá ser orientado a testes.

O teste disciplinado deve seguir os seguintes passos:

1) Produzir um roteiro de teste
    a. definir o contexto (cenário) necessário e selecionar a massa de teste contendo a seqüência de ações e valores de teste com os respectivos resultados esperados e que foi criada segundo um critério de teste.
2) Antes de iniciar o teste: estabelecer o cenário do teste
3) Criar (ou modificar) um módulo controlador de teste, utilizando o *framework* de teste para testar as principais funcionalidades de cada módulo.

4) Desenvolver o código que deve passar nos testes.

5) Ao testar: produzir um laudo em que todas as discrepâncias encontradas são registradas. Esse laudo pode ser uma saída da execução do Gtest ou Catch. Somente termine o teste antes de completar o roteiro, caso observe que não vale mais a pena continuar executando o roteiro, uma vez que o contexto para o resto está danificado.

6) Após a correção: repetir o teste a partir de **2** até o roteiro passar sem encontrar falhas.

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.

Parte 2. Escrita

· Um arquivo **LEIAME.TXT** contendo a explicação de como utilizar o(s) programa(s).
· Tantos arquivos **RELATORIO-nome.TXT** quantos forem os membros do grupo. O tema **nome** deve identificar o membro do grupo ao qual se refere o relatório. Estes arquivos devem conter uma tabela de registro de trabalho organizada como a seguir:
Data | Horas Trabalhadas | Tipo Tarefa | Descrição da Tarefa Realizada

Na descrição da tarefa redija uma explicação breve sobre o que o componente do grupo fez. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. Os tipos de tarefa são:

- estudar aulas e laboratórios relacionados
- especificar os módulos
- especificar as funções
- revisar especificações
- projetar
- fazer diagramas
- revisar projetos
- codificar módulo
- Rodar o verificador estático e retirar warnings
- revisar código do módulo
- redigir casos de teste
- revisar casos de teste

- realizar os testes
- instrumentar verificando a cobertura
- documentar com Doxygen

Observações:

- Dica: Preencha esta tabela de atividades ao longo do processo. NÃO DEIXE PARA ÚLTIMA HORA, POIS VOCÊ NÃO SE LEMBRARÁ DO QUE FEZ TAL DIA, TAL HORA. Com relatórios similares a esse você aprende a planejar o seu trabalho.

Deve ser enviado um único arquivo compactado (.zip) contendo todos os arquivos necessários (ex. .c .h makefile, estrutura de diretório, informação de como utilizar, etc). Deve constar também os documentos especificados. Deverá haver um arquivo leiame.txt com as informações sobre como o programa deverá ser compilado, como poderá ser executado e quais são os arquivos enviados e o que cada um contem.

Apenas um integrante do grupo deve enviar o trabalho. O nome do trabalho deve ser algo como:
MP_Jose_12345_Proj.zip
Com o primeiro nome e matricula de quem envia pelo grupo.

Cópias de trabalho terão nota zero.

Excelente trabalho!

Deve ser enviada pelo ead.unb.br até : 04/07/18 ate as 23:55