Departamento de Ciência da Computação
Métodos de Programação 2/2018

Projeto de Disciplina

Definição dos grupos : 5 / 11 /18

Data de Entrega Especificação: 12/11/18 ate as 23:55

Data de Entrega Projeto: 02/12/18 ate as 23:55

Jogo de Estratégia

Descrição

Deve ser feito um jogo de estratégia que pode ser feito em turnos ou em tempo real (mais detalhes abaixo).

Interface – menu :

O jogo deve ter um menu com pelo menos as opções novo jogo, salvar jogo, carregar o jogo e sair.

Especificação:

No jogo o jogador enfrenta a CPU em um jogo de estratégia. Existe um mapa que mostra todas as informações relevantes.

A temática e elementos do jogo são de livre escolha desde que não tenham conteúdo que possa ser considerado ofensivo.

Devem existir pelo menos dois tipos de recursos necessários para criar as unidades (energia e material por exemplo). A criação de cada unidade requer uma certa quantidade destes recursos. Com o recurso igual a zero é impossível criar novas unidades. Estes recursos devem ser renovados de alguma forma.

Eles podem ser encontrados no mapa ou gerados através de alguma unidade (ex. construir uma usina que gera energia).

Deve ser possível construir pelo menos 3 tipos de prédios. Estes produzem as unidades de combate com uma certa velocidade. Os prédios podem ser destruídos pelos inimigos.

Devem existir pelo menos 3 tipos de unidades de combate. Elas devem ter forças e fraquezas. Por exemplo, a unidade A pode ser forte contra a unidade B e fraca contra a unidade C.

As unidades podem ser movimentadas no mapa e podem destruir outras unidades ou prédios (dependendo das forças e fraquezas). As unidades podem ser eventualmente destruídas. Elas devem ter característica como velocidade, resistência a danos, etc.

O jogo deve ter um critério de finalização como, por exemplo, se passou um tempo determinado, se todas as unidades e prédios foram destruídas, etc.

O jogo pode ser em tempo real ou baseado em turnos.

A CPU deve ter alguma estratégia para ganhar o jogo.

Especificação de requisitos:

Um documento descrevendo a especificação do jogo deve ser enviada até 12/11/18 as 23:55

Envios após esta data terão desconto na nota final do trabalho de 2 pontos

O documento teve conter os integrantes do grupo e uma descrição de qual será o tema do jogo, prédios, unidades, funcionamento, critério de fim de jogo, etc.

O documento deve conter as historietas correspondentes.ao jogo.

As historietas devem ser transformadas em estórias de usuário no formato:

Eu como <tipo de usuario> quero <algum objetivo> para <algum motivo>

Ex. Eu como <jogador> quero <movimentar minhas unidades> para <fugir de um ataque inimigo>

No documento, toda a estória de usuário deverá ter um numero único ex “EU023”. Este número deverá constar no código escrito de forma que se alguém procurar nos códigos pela string “EU023” encontrará o lugar onde a estória foi implementada e o lugar onde ela foi testada.

Estas estórias de usuário devem ser utilizadas no Scrum e Kanban correspondentes.

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.

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 esta 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.

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 utilizando as checklists vistas no trabalho. Deve ser gerado um laudo (pdf) do que passou na revisão e o que não passou. Deve ser mudado o que for possível para estar dentro do especificado na checklist. O que não for possível mudar por

algum motivo forte, deve estar justificado no laudo. Ex. se adequar o código a checklist exigir mudar toda estrutura do programa, não é viável fazer isto.

e) Cada função deve ter assertivas de entrada e de saída como comentários.

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 usando o gcov. Usando o 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 gcov 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.

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)

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 deve 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/

e) **SFML**

Ela pode ser instalada a partir de:

https://www.sfml-dev.org/index.php

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

7) 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)

8) É interessante utilizar o Valgrind (*valgrind.org/*), embora não seja obrigatório.

9) Utilize diretivas de pré-compilação para evitar o problema de identificador duplicado.

10) 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.

Módulos

O programa deve ser dividido em módulos.

O programa deverá ter um makefile adequado colocando os fontes e objetos compilados em diretórios separados como nos trabalhos anteriores.

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.

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:

Definição dos grupos: 5 / 11 /18

Data de Entrega Especificação: 12/11/18 ate as 23:55

Data de Entrega Projeto: 02/12/18 ate as 23:55