Instituto de Matemática e Estatística da USP

MAC0216 - Técnicas de Programação I - 2s2022

EP4
Entrega até 8:00 de 12/12/2022
(INDIVIDUAL)

Prof. Daniel Macêdo Batista

1 Problema

Jogos eletrônicos costumam ter enredos e elementos que conseguem ser facilmente mapeados a princípios de orientação a objetos (OO). Por exemplo, diversos NPCs (*Non-player characters* - personagens não jogáveis) podem existir em um jogo, herdando propriedades de um tipo de personagem genérico mas diferenciando-se entre si em termos de velocidade de movimento, força, habilidades, etc... . O próprio mapa do jogo também pode ser mapeado em princípios de OO já que ele pode possuir em um dado momento diversos personagens no seu espaço. O personagem controlado pelo jogador também pode ser mapeado a princípios de OO, pois pode ser implementado como uma classe que possui diversos atributos (posição no mapa, habilidades disponíveis, nível de vida, etc...), e métodos (caminhar, pular, rastejar, etc...).

Assim como outros sistemas de software, um jogo eletrônico também precisa ser bem documentado e bem testado antes de ser disponibilizado para jogadores e desenvolvedores do mundo todo. E claro, nos casos em que erros sejam detectados na fase de teste, estes devem ser corrigidos antes do lançamento.

A tarefa neste EP é implementar um jogo de Pac-Man[1] mais simples do que o original. A cada iteração, o usuário poderá mover o Pac-Man, enquanto os fantasmas se movem de forma pseudo-aleatória. O jogo deve ser implementado em C++, ter a documentação gerada automaticamente com `doxygen`, a compilação automatizada com o `g++` via `make` e testes unitários realizados com `GoogleTest`.

Este EP vai explorar muito do que foi visto na disciplina, com foco principal na elaboração de testes e na arquitetura do software (principalmente como as classes se conectam). Tudo que foi visto em termos de automação de compilação e de automação de geração da documentação também será exercitada neste EP. Tudo isso em conjunto faz parte do projeto de software e o **gerenciamento de projeto de software** é mais uma importante Técnica de Programação pois auxilia bastante no planejamento, implementação e lançamento de um software.

[1] Se você não faz a mínima ideia de que jogo é este, assista `https://youtu.be/i_OjztdQ8iw`

2 Requisitos

2.1 Funcionamento do jogo

O mapa do jogo, o labirinto, será lido de um arquivo que deverá ser passado como argumento na linha de comando ao executar o EP. O mapa contém as posições iniciais dos fantasmas. O Pac-man sempre inicia no meio do labirinto (os labirintos sempre terão dimensões ímpares e a posição do meio nunca terá uma parede, nem um fantasma e nem um *pac-dot* justamente para garantir que o Pac-man conseguirá ficar no meio).

Cada linha do arquivo representa uma linha do labirinto. Todas as linhas do labirinto devem ter o mesmo número de colunas. Os caracteres usados para representar o labirinto são os seguintes:

- `*` representa uma parede;
- `.` representa um *pac-dot*;
- `F` representa um fantasma;
- Espaço em branco representa uma posição sem qualquer dos elementos acima.

Esses são os únicos caracteres permitidos no arquivo. Fantasmas e paredes são opcionais mas sempre é necessário ter ao menos 1 *pac-dot* e ao menos 1 espaço em branco.

O seu EP deve ser capaz de funcionar com labirintos que tenham qualquer quantidade de linhas, colunas, paredes (`*`), *pac-dots* (`.`), fantasmas (`F`) e espaços em branco.

Abaixo está o conteúdo de um arquivo de labirinto com 31 linhas e 27 colunas. Há 5 fantasmas. Para este labirinto, o Pac-Man deverá ser colocado na linha 15 e na coluna 13[2].

```
***************************
*F.....................F..*
*.**********.**.*********.*
*.**********.**.*********.*
*......**....**...**......*
***.**.**.*******.**.**.***
***.**.**.*******.**.**.***
*...**.................**...*
*.****.*****.**.****.****.*
*.****.*****.**.****.****.*
*............**...........*
******.**.*******.**.******
******.**.*******.**.******
******.**.........**.******
******.**.*******.**.******
******.**.*..  ..*.**.******
..........*.....*.......F..
******.**.*.....*.**.******
******.**.***..**.**.******
******.**.........**.******
******.*****.**.****.******
```

[2]As linhas e as colunas começam a ser contadas do valor zero

```
******.*****.**.****.******
*......**....**...**......*
*.****.**.*******.**.****.*
*.****.**.*******.**.****.*
*.........F..............*
*.****.*****.**.****.****.*
*.****.*****.**.****.****.*
*.****.*****.**.****.****.*
*............**.......F...*
**************************
```

O labirinto terá que ser atualizado constantemente. Sempre que o labirinto for impresso na tela, na posição do Pac-Man deverá ser colocado o caractere `C`. Caso alguma posição não possua nenhum elemento ali, por exemplo porque o Pac-Man comeu o *pac-dot* que estava ali em algum passo anterior, um espaço em branco deve ser colocado no lugar.

Cada iteração do jogo prossegue da seguinte forma:

1. Primeiro, movemos cada fantasma de forma pseudo-aleatória (a cada iteração os fantasmas só podem se mover para uma única direção – cima, esquerda, baixo e direita – e exatamente 1 casa em relação à sua posição atual). Note que dois fantasmas podem ocupar a mesma posição ao mesmo tempo.

2. Verifica-se se houve colisão entre o Pac-Man e algum dos fantasmas, caso em que o jogo termina.

3. O usuário escolhe uma direção de movimento para o Pac-Man e ele é movido uma casa na direção escolhida, se possível. Se a nova casa ocupada pelo Pac-Man contém um *pac-dot*, então o total de *pac-dots* comido é aumentado e aquele *pac-dot* é removido. Se a nova casa ocupada pelo Pac-Man contém um fantasma, então o jogo acaba.

4. Se todos os *pac-dots* foram comidos, então o jogo acaba.

Atenção: No Pac-Man o labirinto é cíclico. Ou seja, se o labirinto tem tamanho 5×7 e o Pac-Man ou um fantasma na posição $(3, 6)$ se move para a direita, então ele reaparece na posição $(3, 0)$, por exemplo. Idem na direção vertical. Desde que não haja uma parede impedindo tal movimento, claro.

Antes de colocar todo mundo para se movimentar é importante mostrar para o usuário uma visão do labirinto para que ele decida para onde ele vai mover o Pac-Man, por isso esse passo tem que ser feito antes do passo que vai movimentar os personagens.

Logo depois de imprimir o labirinto atualizado, o programa deve imprimir um prompt aguardando o movimento do jogador. O prompt precisa ser exatamente assim:

```
Direção (a - esquerda, d - direita, w - cima, s - baixo):
```

Como o prompt sugere, a movimentação do Pac-Man deve ser feita com os caracteres `a`, `d`, `w`, e `s`, que devem ser digitados no teclado seguidos de ENTER. Uma vez lido o caractere, caso a nova posição seja impossível, porque há uma parede ali, o prompt deve ser reimpresso e uma nova posição deve ser informada pelo jogador.

Antes do usuário ter outra chance de mover o Pac-Man, após ter movido para uma posição válida, todos os fantasmas devem andar 1 casa para alguma direção de forma pseudo-aleatória.

Sempre que os fantasmas ou o Pac-Man se movimentam, pode ser que o jogo termine por causa do choque de algum fantasma com o Pac-Man. Caso tenha havido algum choque, o programa deve imprimir o labirinto com um `X` na posição onde houve o choque e terminar com a mensagem abaixo:

```
Game over! Pontos = %d
```

onde o %d representa quantos *pac-dots* foram comidos pelo Pac-Man. Enquanto não houver choque e enquanto ainda houver *pac-dots*, o programa deve continuar.

Sempre que o Pac-Man se movimentar e não se chocar com nenhum fantasma, se havia um *pac-dot* na posição para onde ele se moveu, remove-se o *pac-dot* do labirinto, colocando uma posição vazia no lugar, e aumenta-se a pontuação do Pac-Man de 1 unidade.

Após as movimentações dos personagens, se o jogo não terminar por choque do Pac-Man com algum fantasma, pode ser que o Pac-Man tenha comido o último *pac-dot*. Nesse caso o programa deve imprimir o labirinto e terminar com a mensagem:

```
Congratulations! Pontos = %d
```

2.2 *Headers*

Todos os arquivos `.cpp` e `.h` do seu EP só poderão ter no máximo estes includes:

```
#include <iostream>
#include <fstream>
#include <random>
#include <gtest/gtest.h>
```

além é claro de incluírem uns aos outros quando necessário.

2.3 Classes

Seu programa precisará ter exatamente estas classes abaixo. Os nomes delas são suficientes para você entender como elas devem se integrar:

- Arquivo
- Labirinto
- Personagem
- Fantasma
- Pacman
- Partida

2.4 Documentação

Os comentários nos códigos-fonte explicando as classes e o programa principal precisarão ser suficientes para que o programa `doxygen` possa gerar uma documentação do software em arquivos html e demais formatos necessários para que a documentação possa ser lida em um navegador web. Certifique-se que a documentação seja gerada corretamente com o arquivo de configuração do `doxygen` da aula 19 com apenas esta diferença:

```
OPTIMIZE_OUTPUT_FOR_C = NO
```

Além é claro da linha `INPUT` = que precisará apontar para os arquivos deste EP.

2.5 Compilação

O código principal precisa ser compilado apenas com o comando `make`. O `Makefile` incluído na entrega precisa ser tal que tenha pelo menos estes alvos:

- `all`: que vai compilar o programa sem os testes ativados
- `doc`: que vai gerar a documentação
- `tests`: que vai compilar o programa ativando todos os testes
- `clean`: que vai limpar todos os arquivos gerados na compilação e na geração da documentação

2.6 Testes

Para conferir que todos os métodos das classes estão corretamente implementados você deve escrever testes unitários, usando `GoogleTest`[3]. Os testes devem ser escritos no arquivo `ep4.cpp` e deve ser ativados ao compilar o código com `make tests`.

2.7 LEIAME

Junto com o código-fonte do programa você terá que entregar também um arquivo LEIAME em texto puro (Ele pode ser nomeado apenas como LEIAME ou pode ser LEIAME.txt). Crie o seu arquivo com no mínimo estas cinco seções dentro dele:

```
AUTOR:
<Seu nome, NUSP e endereço de e-mail>

DESCRIÇÃO:
<Explique o que o programa faz (Não diga apenas que ele é o EP4 da
disciplina tal. Explique o que de fato o programa faz)>

COMO EXECUTAR:
<Informe como compilar o programa e como executá-lo. Você pode
informar que basta executar o comando make mas é importante explicar
sobre o argumento de linha de comano do programa (o que é esse
argumento, como ele precisa estar formatado, etc...)>

TESTES:
<Forneça explicações sobre os testes embutidos na sua entrega>

DEPENDÊNCIAS:
<Informe o que é necessário para compilar e rodar o programa.
Informações como o SO onde você executou e sabe que ele funciona são
importantes de serem colocadas aqui.>
```

[3] https://google.github.io/googletest/primer.html

3 Entrega

Você deverá entregar um arquivo tarball comprimido (.tar.gz) contendo os seguintes itens:

- Interface e implementação de todas as classes:
  - `.cpp` e `.h` da classe Arquivo;
  - `.cpp` e `.h` da classe Labirinto;
  - `.cpp` e `.h` da classe Personagem;
  - `.cpp` e `.h` da classe Fantasma;
  - `.cpp` e `.h` da classe Pacman;
  - `.cpp` e `.h` da classe Partida;
- 1 código principal `ep4.cpp` com o main do EP e os testes;
- 1 Makefile
- 1 arquivo LEIAME (pode ser LEIAME.txt) em texto puro;

Todos esses arquivos devem estar presentes na raiz da pasta do EP (Não use subdiretórios!). Caso algum arquivo esteja faltando, o EP não será corrigido e a nota dele será zero.

O desempacotamento do tarball deve produzir um diretório contendo os itens. O nome do diretório deve ser ep4-seu_nome. Por exemplo: ep4-joao_dos_santos.

A entrega do tarball deve ser feita no e-Disciplinas.

O EP deve ser feito individualmente.

Obs.1: Serão descontados 2,0 pontos de EPs com tarballs que não estejam nomeados como solicitado ou que não criem o diretório com o nome correto após serem descompactados. Confirme que o seu tarball está correto, descompactando ele no shell (não confie em interfaces gráficas na hora de verificar o seu tarball pois alguns gerenciadores de arquivos criam o diretório automaticamente mesmo quando esse diretório não existe).

Obs.2: A depender da qualidade do conteúdo que for entregue, o EP pode ser considerado como não entregue, implicando em MF=0,0. Isso acontecerá também se for enviado um tarball corrompido, códigos fonte vazios ou códigos fonte que resolvam um problema completamente diferente do que foi solicitado.

Obs.3: O prazo de entrega expira às 8:00:00 do dia 12/12/2022.

4 Avaliação

70% da nota será dada pela implementação, 10% pelo LEIAME e 20% pelos testes. Os critérios detalhados da correção serão disponibilizados apenas quando as notas forem liberadas.