ACH2003 - Computação Orientada a Objetos - 2020

EP2 - Pong! 2.0

Descrição

Após o sucesso de **Pong!**, lançado pela *EACH Game Dev. Co.* e finalizado graças a sua ajuda, cabe a você a tarefa de implementar alguns recursos novos a serem incorporados na nova versão do jogo (**Pong! 2.0**). Mais especificamente, sua tarefa consiste em **implementar uma nova classe** e **completar uma classe já existente**.

A nova classe que deve ser implementada é a **FxBall**. Seu papel é fornecer uma nova implementação de bola que acrescenta um efeito visual de *rastro* (veja o vídeo atualizado do *gameplay* para entender melhor do que se trata). A nova classe deve, obrigatoriamente, implementar a interface **IBall**. Além disso, você deve reaproveitar a implementação já existente na classe **Ball,** seja através de herança, ou seja através de composição. Ao reaproveitar o que já existe na classe **Ball**, evita-se a reimplementação dos comportamentos básicos da bola como: movimentação, verificação de colisão, e alteração do movimento da bola devido a colisões (a propósito, o forma como a bola rebate nos players foi alterada para deixar o jogo menos previsível e mais dinâmico). Esta classe deve, **obrigatoriamente**, possuir um construtor com assinatura idêntica à do construtor da classe **Ball**.

Já a classe que deve ser completada é a classe **BallManager**. Com a previsão de que, em determinadas situações (mais detalhes a seguir), poderá haver mais de uma bola em jogo, esta nova classe foi projetada para assumir o papel de gerenciar todas as bolas ativas em uma partida. Uma implementação básica desta classe (que gerencia apenas uma bola - a bola principal do jogo) foi disponibilizada a você, para que você possa completá-la de acordo com os novos recursos que devem ser implementados. Com a introdução da classe **BallManager**, a classe **Pong** não mais gerencia diretamente a(s) bola(s) em jogo, mas faz isso através do intermédio desta nova classe.

Além de gerenciar a eventual existência de múltiplas instâncias de bola, você também precisa completar o código da classe **BallManager** de modo que ela seja responsável por gerenciar a interação da(s) bola(s) com um tipo novo de elemento presente no jogo: os *targets* (alvos). Na versão 2.0 do jogo, existem dois tipos de alvo: **DuplicatorTarget** e **BoostTarget**, ambos derivados da classe mãe **Target**. Os alvos são similares às paredes (instâncias da classe **Wall**), mas ao invés de rebater a bola, provocam algum efeito quando atingidos, com cada tipo de alvo causando um efeito diferente.

Quando um alvo do tipo **BoostTarget** é atingido por uma bola, a bola deve ter sua velocidade multiplicada pelo fator definido na constante **BOOST_FACTOR** (declarada na própria classe **BoostTarget**) e tal efeito deve durar pelo intervalo de tempo (em

milisegundos) definido na constante **BOOST_DURATION** (também declarada em **BoostTarget**). Passado o intervalo de tempo, a velocidade deve ser restaurada para a velocidade padrão. Se uma bola, já com a velocidade aumentada atingir novamente um **BoostTarget**, nenhuma ação deve ser realizada.

Já quando um alvo do tipo **DuplicatorTarget** é atingido por uma bola, uma nova bola deverá ser adicionada ao jogo. A nova bola deverá surgir na mesma posição da bola que atingiu o alvo, mas com uma nova direção (que deve ser determinada pela soma de uma perturbação aleatória à direção da bola que atingiu o alvo). A nova bola deverá possuir a velocidade padrão, não importando se a bola que colidiu com o **DuplicatorTarget**, está (ou não) sob o efeito de ter atingido um **BoostTarget** previamente. Além disso, a nova bola deve ter cor diferente da bola principal (aquela que existe desde o início da partida), e deve expirar após o intervalo de tempo (em milisegundos) definido pela constante **EXTRA_BALL_DURATION** (na própria classe **DuplicatorTarget)**. A nova bola, com exceção da cor diferente e do fato de expirar após um intervalo de tempo, deve se comportar exatamente como a bola principal do jogo em relação ao seu comportamento de movimento e pontuação.

Note que as classes **BoostTarget** e **DuplicatorTarget** já se encontram prontas, e não devem ser alteradas. Estas duas classes são, em essência, muito parecidas (a rigor, apenas a superclasse **Target** seria necessária), mas o fato de existirem dois tipos distintos poderá ser conveniente para determinar com qual tipo de alvo uma bola colidiu. Observe ainda que toda interação entre as bolas e os alvos, bem como a aplicação e gerenciamento dos efeitos, devem ficar sob responsabilidade da classe **BallManager**.

Em resumo, na sua versão 2.0, o projeto **Pong!** é composto pelas seguintes classes:

- **Pong**: classe principal do jogo (contém o método `main` que gerencia o todo andamento da partida e a interação entre os demais elementos).
- **GameLib/MyFrame/MyKeyAdapter**: três classes que implementam funcionalidades gráficas (criação de janela em modo gráfico, métodos para desenhos de formas geométricas) e para processar entrada via teclado.
- **IBall**: declara a interface comum a todas as bolas que venham a ser implementadas.
- **Ball**: implementa a bola “padrão” do jogo.
- **Player**: implementa os jogadores (controláveis) pelo usuários.
- **Wall**: implementa as paredes do jogo.
- **Score**: implementa o placar do jogo.
- **Target/BoostTarget/DuplicatorTarget**: implementação dos alvos que produzem algum tipo de efeito no jogo, quando atingidos por uma bola.
- **BallManager**: gerencia uma ou mais bolas presentes no jogo, assim como a interação das mesmas com os alvos e a aplicação dos efeitos correspondentes.

Destas, apenas o código fonte da classe **BallManager** precisa ser completado. Você também deve criar a classe **FxBall** (respeitando as restrições já especificadas para ela) e pode, eventualmente, criar novas classes que julgar convenientes. A única restrição é que

não são permitidas alterações das assinaturas dos métodos já existentes na classe **BallManager**.

Adicionalmente, também é fornecido o código fonte da classe **Pong**, para que vocês possam ter uma visão geral de como o jogo é implementado, mas alterações não devem ser feitas nesta classe pois, na correção, nossa própria versão da classe será usada. Desta forma qualquer alteração feita por você na classe **Pong** não será considerada.

Compilando e executando o Pong! 2.0

Para compilar o jogo basta executar, no terminal, o comando:

```
$ javac *.java
```

Já para rodar o jogo em sua versão 2.0, há alguns parâmetros opcionais que podem ser especificados. Tais parâmetros são recebidos pela linha de comando e, apesar de serem opcionais, devem ser sempre passados na mesma ordem, quando usados (por exemplo, se você quiser especificar o segundo parâmetro, deve obrigatoriamente especificar o primeiro também). A linha de comando para rodar o jogo deve seguir o seguinte padrão:

```
$ java Pong <classe_bola> <intervalo_de_tempo> <safe_mode>
```

Através do parâmetro `classe_bola` pode-se especificar qual o tipo das bolas que serão criadas pela classe **BallManager**. Se não for especificado (ou se for especificado um nome de classe inválido), o valor padrão padrão que será considerado para este atributo será “Ball”, correspondente à implementação “padrão” da bola (classe **Ball**). Depois que você tiver implementado a classe **FxBall**, você deverá executar o jogo da seguinte forma para fazer com que o jogo use a nova implementação:

```
$ java Pong FxBall
```

Qualquer nome de classe pode ser usado, desde que a classe em questão implemente a interface **IBall** e possua um construtor com assinatura idêntica à do construtor de **Ball**. Para experimentar melhor este funcionamento (antes mesmo de implementar a classe **FxBall**), estude e compile a classe **DiamondBall** (cujo código também é fornecido), que redefine o comportamento de desenho da bola “padrão”, e em seguida execute o jogo da seguinte forma:

```
$ java Pong DiamondBall
```

O bacana deste mecanismo (que instancia os objetos a partir de um nome de classe) é que todo o código do jogo fica sem relações de dependência com as novas implementações de bola que venham a ser criadas. Desta forma a criação de novos tipos de bola não exige a recompilação de todo o código do jogo.

Em relação ao parâmetro `intervalo_de_tempo`, pode-se usá-lo para especificar o intervalo de tempo mínimo (em milissegundos) que deve ser aguardado entre o processamento de dois frames consecutivos do jogo. Este parâmetro possui valor padrão igual a 3 e, em geral, não há razão para alterá-lo.

Por fim, o parâmetro `safe_mode`, quando definido como `true`, ativa o modo de segurança do modo gráfico implementado pela classe **GameLib**. Este parâmetro possui valor padrão igual a `false`, e deve ser usado caso a tela do jogo não seja exibida de forma correta usando o modo padrão (que é mais eficiente quando funciona adequadamente).

A linha abaixo ilustra como executar o jogo especificando todos os 3 parâmetros opcionais (no caso, para usar a bola implementada pela classe **DiamondBall**, determinar o intervalo de tempo entre dois frames como 5 milissegundos e habilitar o safe mode):

```
$ java Pong DiamondBall 5 true
```

Entrega

Este Exercício-Programa é individual e deve ser entregue na tarefa disponível no portal eDisciplinas até o dia 13 de junho de 2020.

Você deve entregar um arquivo compactado nos formatos `.zip` ou `.tar.xz` contendo:

- um arquivo texto `LEIAME.txt`, com instruções detalhadas para a **compilação** e **execução** do programa (em linha de comando; não dar instruções que envolvam o uso de alguma IDE específica). Encorajamos (mas não se sintam obrigados) a fazer o uso de uma ferramenta que facilite a compilação de um projeto como o GNU Make [1] (mais simples, amplamente adotada pela comunidade de desenvolvimento de código livre e a ferramenta usada para compilar, por exemplo, o código do próprio Linux) ou o Gradle[2] (um pouco mais complicada, uma ferramenta utilizada pela comunidade de desenvolvimento Java e adotada, por exemplo, pela comunidade Android).
- um diretório `src`, com todos os arquivos que fazem parte do código-fonte do programa e que são necessários para compilar o programa.

Ambiente de correção

[1] GNU Make: https://www.gnu.org/software/make/
[2] Gradle: https://gradle.org/

O Exercício-Programa será avaliado em um computador equipado com o **sistema operacional Linux** e com os seguintes softwares e versões:

```
$ java --version
openjdk version "14.0.1" 2020-04-14
OpenJDK Runtime Environment (build 14.0.1+7-Debian-1)
OpenJDK 64-Bit Server VM (build 14.0.1+7-Debian-1, mixed mode,
sharing)

$ make --version
GNU Make 4.2.1
Compilado para x86_64-pc-linux-gnu
Copyright (C) 1988-2016 Free Software Foundation, Inc.

$ gradle -version
------------------------------------------------------------
Gradle 6.4
------------------------------------------------------------

Build time:   2020-05-05 19:18:55 UTC
Revision:  42f7c3d0c3066b7b38bd0726760d4881e86fd19f

Kotlin:    1.3.71
Groovy:    2.5.10
Ant:       Apache Ant(TM) version 1.10.7 compiled on September 1
2019
JVM:       14.0.1 (Debian 14.0.1+7-Debian-1)
OS:        Linux 5.6.0-1-amd64 amd64
```

Have fun! :)