Pacman: uma implementação em Java

11030416 - Igor Neres Trindade

1 de Agosto de 2018

Turma: A1 - Matutino

1 Introdução

Neste projeto, desenvolvido para a matéria de Programação Orientada a Objetos, ministrada pelo Prof. Dr. Paulo Pisani na Universidade Federal do ABC, implementamos o clássico jogo Pacman na linguagem Java.

Tivemos, como meta principal, a implementação das mecânicas básicas do jogo original, isto é, a criação de diferentes mapas, nos quais fanstamas se movimentam com uma dada inteligência artificial em busca do Pacman, controlado pelo usuário, com o objetivo de ingerir todas as comidas do mapa. T

Para complementar, desenvolvemos o jogos com uma interface gráfica, através da biblioteca JavaFx.

2 Descrição das classes e interfaces

Ao todo, 17 classes foram implementadas de forma a manter baixo acoplamento e alta coesão. A seguir, há uma breve descrição de cada uma delas. Suas relações estão representadas no Diagrama de Classes (Figura 1)

- Game

  Classe principal do jogo. Possui o método estático main(), por onde o programa é iniciado. Nele, através de um método herdado da classe Application (importada pela biblioteca do JavaFx), é criada uma nova instância dessa mesma classe, seguida da chamada do método start() sobrescrito.

  Uma vez criada essa instância e chamado o método start(), uma instância de ScreenSetting é criada (e isso ocorre através do padrão de projeto Singleton). O Stage é passado como argumento e modificado somente no momento da primeira instanciação. As novas instâncias de Scene, que representam os conteúdos físicos de uma aplicação em JavaFx e são adicionados à instância de Stage em ScreenSetting para mostrar interfaces gráficas diferentes, são trocadas nas outras classes, que acessam essa instância criada - justamente por esse padrão Singleton.

- ScreenSetting

  Classe que guarda um objeto do tipo Stage que é instanciado uma única vez. Feita essa primeira instanciação, todas as outras instâncias de Scene criadas serão adicionadas a ela. A classe ScreenSetting também possui informações importantes referentes à interface gŕafica criada, como largura e altura da tela. Possui métodos setMenu() e setGameplay() que instanciam novos objetos das classes MenuScene() e GameScene(), respectivamente, e acessam as suas instâncias de Scene já devidamente configuradas.

  Possui o padrão Singleton. Desta forma, a instância criada na classe Game pode ser acessada em quaisquer outras classes.

- IScene

  Interface criada para uniformizar as classes que criam instâncias de Scene já devidamente configuradas, como o MenuScene e GameScene. Devemos assegurar que essas classes tenham um método getScene(), para retornarem uma instância da classe Scene a ser adicionada à instância de Stage padrão, que se encontra na instância única de ScreenSetting.

- MenuScene

  Classe que, como implementa a interface IScene, tem um método getScene() e, para tal, possui um objeto da classe Scene. Este será devidamente configurado. Essa classe também implementa EventHandler<KeyEvent> e sobrescreve seu método handle(), no qual são descritas as ações a serem tomadas para cada evento do teclado.

- GameScene

  Classe que implementa a interface IScene e EventHandler<KeyEvent>, assim como o MenuScene. Adicionalmente, ele possui todos os objetos necessários para as mecânicas do jogo, criando instâncias do Pacman, dos quatro outros fantasmas e do mapa de acordo com a seleção do usuário. Vale lembrar que seu handle() sobrescrito é diferente do MenuScene, especifico para a movimentação do personagem pelo usuário.

- Player

  Classe abstrata que representa um jogador, seja fantasma ou pacman. Não pode ser instanciada, por ser abstrata, e seus métodos públicos e protected são herdados pelas classe que a estendem. Nela, temos atributos que são comuns a todos os jogadores no mapa: o posicionamento, a imagem de cada personagem (lembremos que estamos trabalhando com uma interface gráfica) e o tipo de movimento.

  O método principal para movimentação do jogador, move(), é abstrato. Assim, ele obrigatoriamente teve que ser implementado nas classes que estendem Player.

- Ghost

Classe que, assim como Player, é abstrata. Possui métodos importantes que são compartilhados com as quatro classes instanciáveis de fantasma, entre eles, a movimentação aleatória. Quando o método move() é chamado, a escolha da movimentação é feita. Esta escolha, para não resultar em uma movimentação travada, ocorre considerando os caminhos possíveis, com exceção do bloco anterior. O bloco anterior entra como possibilidade somente no caso de o fantasma ficar travado com as três primeiras escolhas.

- BlueGhost, RedGhost, YellowGhost e GreenGhost

  Classes instanciáveis de fantasmas. A inteligência artificial foi implementada na classe Ghost por métodos que são herdados.

  Houve uma necessidade de implementar uma classe para cada fanstamas para, no construtor, passar o diretório de cada imagem para a classe-mãe. Ademais, para futuras melhorias, seria possível implementar estratégias diferentes para cada fantasma, especialmente utilizando o padrão de projeto Strategy

- Pacman

  Classe que representa o personagem principal do jogo, Pacman, controlado pelo usuário. Como o movimento ocorre de acordo com o usuário, nessa classe há diversos métodos que verificam a possibilidade desse movimento, e verificam quando o Pacman toca algum fantasma, que resulta no fim do jogo. Para tal, tivemos que declarar um atributo privado para monitorr se esse jogador morreu ou não.

- Map

  Classe que representa o mapa no qual os personagens se movimentam. Possui como principais atributos um vetor de objetos da classe Tile, que representam cada bloco do mapa, e a quantidade de comida existente no mapa. Quando o Pacman, controlado pelo usuário, ingerir todos os existentes no mapa, o jogo termina.

- Tile

  Classe abstrata que representa um bloco do mapa. Seu principal atributo, compartilhado com todas as classes que a estendem, é a imagem do bloco (objeto instanciado da classe ImageView).

- Brick

  Classe que representa um tijolo no mapa, ou seja, um bloco inacessível para os jogadores. Não possui quaisquer outros métodos relevantes. Seu método construtor passa como argumento para o método construtor da classe-mãe o diretório da imagem de um tijolo.

- Path

Classe que representa um caminho para os jogadores. Pode ou não ter comida e, desta forma, possui um atributo privado para tal, assim como métodos para verificar se há comida e para limpá-la da comida. Seu método construtor passa como argumento para o método construtor da classe-mãe o diretório da imagem de um caminho.

- ScreenException

  Classe que estende uma Exceção. Usado para o tratamento de exceções, principalmene no que diz respeito ao não carregamento correto das images. Quando isso ocorrer, seja pela inexistencia dessa arquivo no diretório correto, uma mensagem de erro é impressa e o jogo volta para a tela de Menu.

3 Conceitos de orientação a objetos aplicados

Todos os conceitos de orientação a objetos vistos em aula foram aplicados.

Na modelagem do projeto, tivemos o devido cuidado com encapsulamento, declarando todos atributos de forma a serem acessíveis somente na instância da classe e, quando se fez necessário seu uso em outras clases, utilizamos métodos get/set.

Além disso, modelamos de forma a ter o mínimo de acoplamento com o máximo de coesão, isto é, tentamos deixar o código flexível, reusável e organizado, contendo métodos com poucas responsibilidades e com focos em pequenos problemas específicos.

Utilizamos herança diversas vezes: Game estende a classe Application, Ghost e Pacman estendem Player, Path e Brick estendem Tile, ScreenException estende Exception. Para esses casos de herança, sobrescrevemos alguns métodos. Citemos, por exemplo, o método move() que é abstrato na classe Player e é implementado em Ghost e em Pacman.

Para todas as classes criadas, utilizamos construtores não-default. Também utilizamos sobrecarga de método, como se pode ver na classe ScreenSetting, no construtor.

Por fim, foi criada uma interface para manter um padrão referente às classes que geram a instância de Scene a ser adicionado ao Stage principal de ScreenSetting. Utilizamos tratamento de exceções também. As exceções podem ser lançadas durante a criação de uma instância de ImageView para cada elemento do jogo. O tratamento é feito nas classes que implementam a interface IScene, ou seja, nas classes MenuScene e GameScene. O seu tratamento, além de mostrar uma mensagem especificando o tipo de erro, evita que o jogador avance nas outras tela do jogo, deixando-o no Menu.

Vale citar também que foi utilizado um padrão de projeto: Singleton. Ele possibilitou a criação de uma instância única da classe ScreenSetting. Tal padrão possibilitou o acesso a essa instância em outras classes, adicionando diferentes instâncias de Scene à instância única de Stage em ScreenSetting.

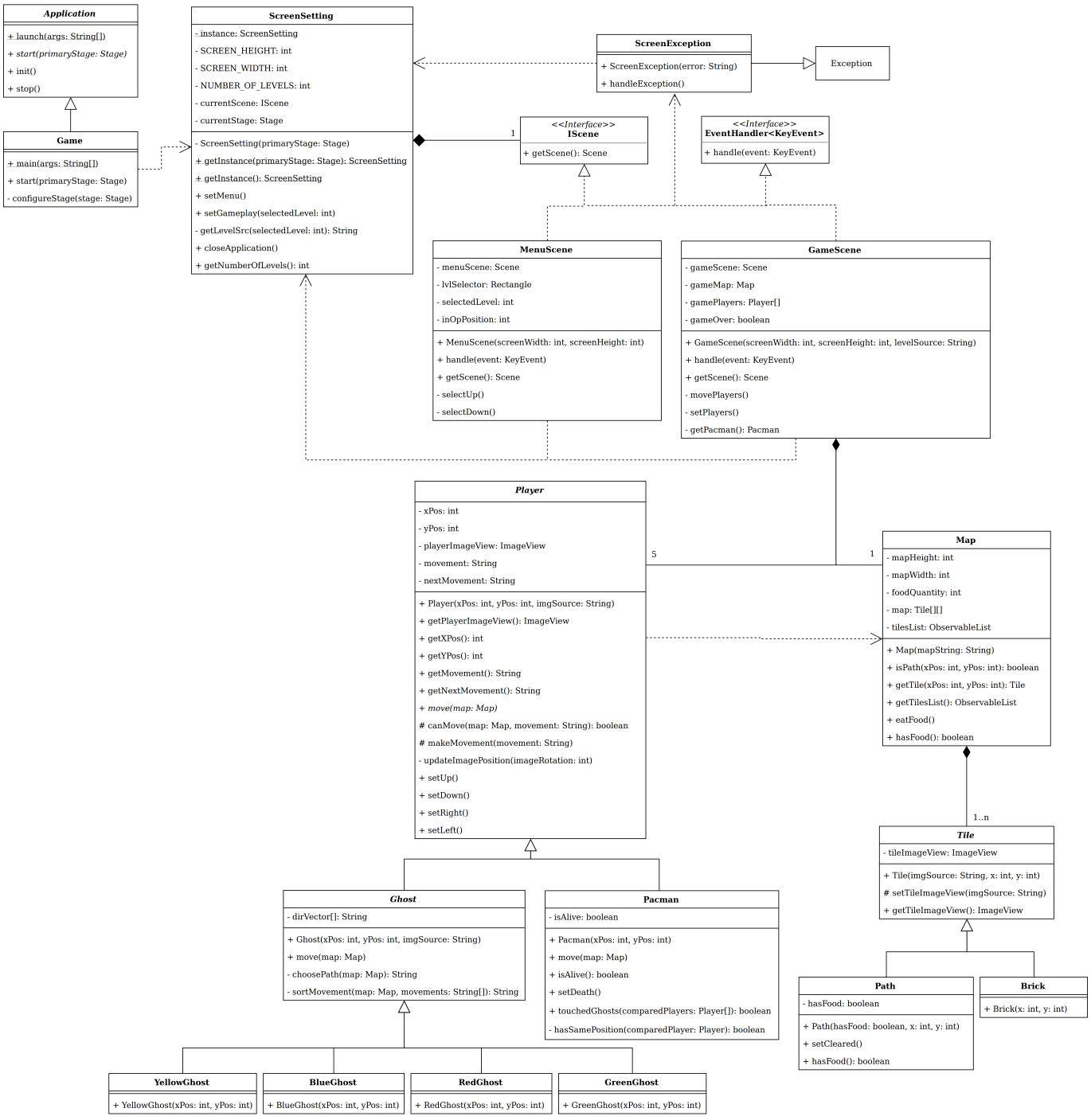

Figura 1: Diagrama de classes (UML), representando seus atributos e a relação entre as classes