Projeto 2 - Pac Man

Programação Orientada a Objetos

Bernardo Souza Muniz e Ygor Martins.

03 de Julho de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução

Este relatório tem o objetivo de apresetar a análise técnica e o diagnóstico do código proposto ao Projeto PM (Pac Man). Será abordado uma descrção geral das funcionalidades presentes atualmente no código, os problemas relacionados aos princípios de POO (***Programação Orientada a objetos***) bem como uma proposta de reorganização do código baseada em diagramação UML (***Unified Modeling Language***).

2. Descrição geral das funcionalidades

A por cada método presente nas classes *App* e *PacManGame* com o intuito de entender a fundo cada funcionalidade implementada.

Em primeira vista, ao fazer uma análise básica da organização de pastas e de classes do projeto original do jogo, é possível verificar que há somente dois arquivos que fazem toda a separação de classes do jogo, sendo eles: `App.java` e `PacMan.java`. Nota-se que não há uma organização prévia de classes, separação de responsabilidades e funcionalidades específicas de cada componente do Jogo.

A classe mais externa `PacMan.java` faz a implementação das interfaces `ActionListener` e `KeyListener`. Ambas interfaces são responsáveis pela identificação de teclas apertadas no teclado e mouse pelo usuário no controle do PacMan. Além disso, há uma herança com a classe `JPanel`, que é responsável pela criação das interfaces gráficas do jogo.

Todos os objetos presentes no jogo, sendo eles: paredes, fantasmas de variadas cores, pontos de jogo e o próprio PacMan fazem parte de uma classe mais interna denominada `Block`. Essa classe encapsula os atributos comuns a todos os elementos presente no mapa, como posição xy, largura, altura, imagem, direção, posições iniciais e velocidades xy.

Embora o atributo de velocidade contemple todos os blocos, é interessante destacar que existem componentes do mapa que permanecem estáticos durante todo o jogo, como a parede e os ponto de jogo. Baseado nisso, é possível verificar que apesar de compartilharem a mesma estrutura base, o comportamento de cada objeto varia durante a execução do jogo.

O mapa do jogo é carregado através de um vetor de strings privada na classe PacMan. Foi utilizado uma nomenclatura padronizada para cada elemento do mapa quando o jogo é carregado. Sendo: X (paredes),“ “ (Foods), “O” (Nada), “P” (PacMan), “b” (Fantasma Azul), “r” (Fantasma Vermelho), “p” (Fantasma Rosa), “o” (Fantasma Laranja).

Para as funcionalidades de movimentação, foram utilizadas funções que são responsáveis por aumentar a velocidade do bloco com base na sua direção e de atualizar a direção do bloco do bloco caso haja uma colisão com a parede.

3. Identificação e explicação dos problemas de POO

Os principais problemas de POO **(Programação orientada a objetos)** que foram identificados durante a reestruturação do projeto são listados abaixo:

3.1. Criação de novos objetos

Um dos principais problemas de POO que foram identificados no projeto foi a falta de criação de novas classes para a separação de responsabilidades no código. A classe PacMan faz a instanciação de todos os componentes presentes no jogo, da renderização do mapa e lida com toda a lógica do jogo. Não se tem uma estruturação de classes que permita separar e definir o que cada classe faz no jogo.

Essa abordagem torna-se inviável a longo prazo, principalmente se houver a necessidade de realizar futuras atualizações ou adições de funcionalidades. Com toda a lógica centralizada em uma única classe, a manutenção do código se torna difícil e com grandes chances de erro.

3.2. Falta de herança entre as classes

Todo novo componente do mapa é baseado na classe `Block`, ou seja, a classe bloco define os atributos e comportamentos de cada objeto do jogo. O principal problema dessa modelagem de classes é que ela não define de forma clara quais são os diferentes tipos de objetos do jogo.

Por exemplo, os pontos de jogo que o PacMan come é considerado um bloco e seu score é definido durante incrementado durante a execução do jogo. Seria muito mais legível criar uma classe de objetos comestíveis que tem no seu método construtor um score inicial a ser definido.

3.3. Falta de separação de atributos

Como a classe `Block` faz a instância de todo componente do mapa, ela oferece atributos e métodos que não são convenientes para alguns componentes do mapa. Por exemplo, as paredes e os pontos de jogo não precisam de um método para atualizar a velocidade e atualizar a direção, pois ficam estáticos durante todo o jogo.

O ideal seria criar uma classe que defina e separe os métodos e atributos de um componente que precisa se mover e atualizar sua direção.

3.4. Falta de interfaces e classes abstratas

A ausência de interfaces e classes abstratas, bem como a falta de métodos sobrescritos que definem o comportamento específico de cada tipo de objeto, dificulta a legibilidade, reutilização e extensão do código.

Como todos os elementos do jogo são representados pela mesma classe (Block), não é possível especializar comportamentos de forma clara e organizada. Por exemplo, seria

interessante criar uma classe que define o comportamento de comer de cada objeto que for comestível dentro do jogo.

4. Proposta de reorganização com diagrama de classes

Em vista dos conceitos apontado, foi montado o seguinte diagrama UML:

Figura 1: Elaborada pelos autores

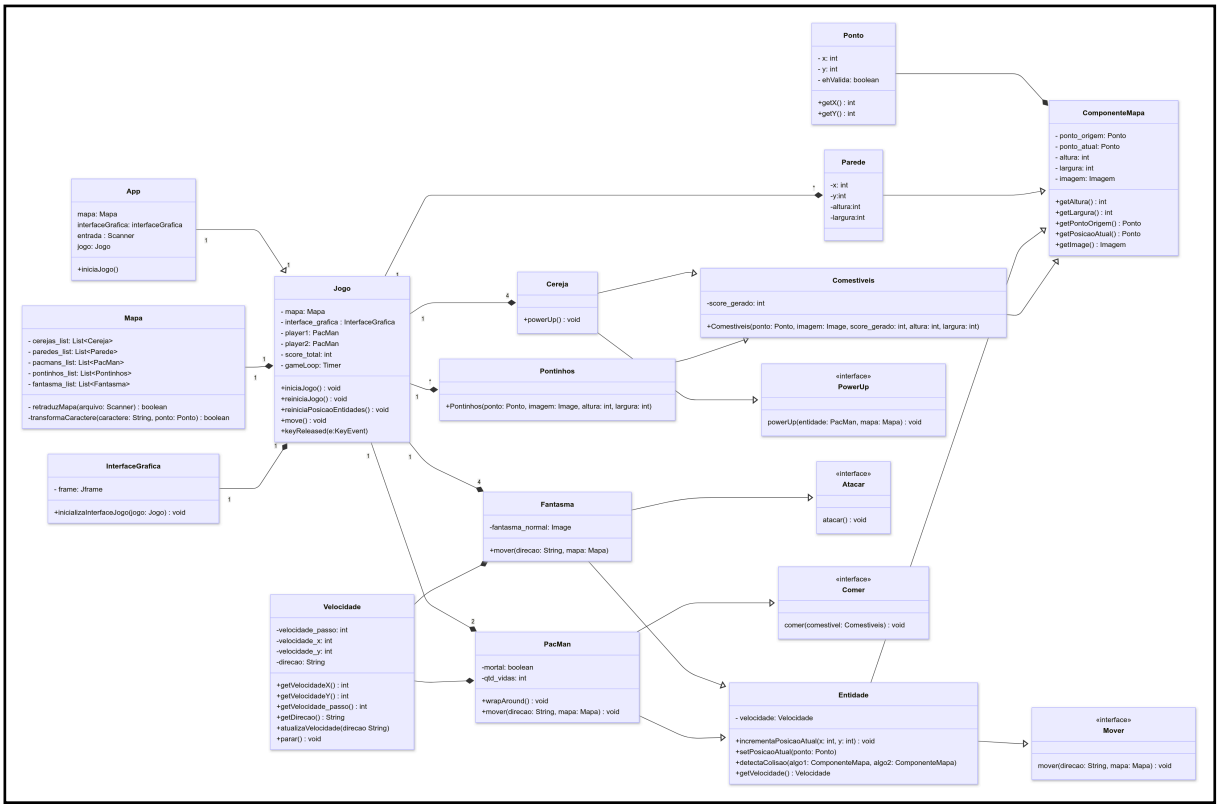

Diagrama UML

5. Conclusão

Podemos perceber que o código apresentado fere diversos princípios da programação orientada a objetos, tal como sua base que são os princípios SOF. Fazendo o uso escasso de classes, e sem utilizar conceitos como herança, polimorfismo, classes abstratas, e por fim, interfaces.

A partir deste problema, fomos capazes de remodelar o código aplicando conceitos como de POO, com o fim de tornar este mais eficiente, legível, e principalmente, mais flexível a alterações futuras. Fazendo assim com que sua durabilidade e qualidade fossem preservadas de maneira eficiente.