

Universidade Federal de Juiz de Fora

DCC168 - Teste de Software

Daniel Rezende Varoto  
Julio Cesar Rosa Trindade

Sumário

[1- Introdução 3](#_Toc26220388)

[2 - Ferramentas 3](#_Toc26220389)

[3 - Teste Funcional 3](#_Toc26220390)

[4 - Teste Estrutural 6](#_Toc26220391)

[5 - Teste de mutação 12](#_Toc26220392)

[6 - Referências 13](#_Toc26220393)

# 1- Introdução

Dentro do processo de desenvolvimento de software, a qualidade e funcionamento do software são de suma importância para o projeto como um todo. Tendo em vista os diversos aspectos qualitativos de um software, o processo de testes de software prevê métodos de testes para cada uma das partes do software, indo desde testes funcionais, em que a interface com o usuário é explorada até testes estruturais, que avaliam a construção e funcionamento de componentes internos não vistos diretamente pelo usuário final.

A fim de exercitar estas metodologias, apresentamos o trabalho final da disciplina DCC168 – Testes de Softwares.

Para este trabalho, escolhemos o jogo PongGame[1] para aplicação de testes funcionais, estruturais e baseados em defeito, conforme proposto no documento de especificação do trabalho

# 2 - Ferramentas

Para análises e implementações, foram utilizados os seguintes softwares:

1. Eclipse IDE [2]
2. JUnit [3]
3. PIT Mutations [4]

# 3 - Teste Funcional

Para aplicação dos testes funcionais, foi necessário que fizéssemos uma análise exploratória no jogo, já que o mesmo não possuía documentação.

Dentro dessa análise, identificamos as seguintes funcionalidades:

1. Start: Onde o jogo acontece
2. Help: Tela com instruções do jogo
3. Quit: Sair do jogo
4. Tela de FPS: Tela que mostra quantidade de FPS (Frame per second) que o jogo exibe

De posse dessa informação, efetuamos os testes funcionais, conforme previsto.

Primeiramente, executamos a técnica de particionamento por classe de equivalência.

Dessa forma, conseguimos gerar todas as possíveis entradas separando os testes em casos que são válidos (V) e inválidos (I), conforme pode ser visto nas tabelas abaixo.

|  |  |  |
| --- | --- | --- |
| Menu Principal | | |
| **Variáveis de entrada** | **Saída esperada** | **Classe de Eq. Cobertas** |
| Seta para baixo e ENTER | Entrada na opção HELP | V1, V2 |
| Pressionar F | Entrada na tela de FPS | V3 |
| Clicar sobre a opção HELP | --- | I5 |
| Pressionar 0 | --- | I1 |
| Pressionar ! | --- | I2 |
| Pressionar A | --- | I3 |
| Pressionar z | --- | I4 |

Tabela 1- Particionamento por classes de equivalência para tela Menu Principal

|  |  |  |
| --- | --- | --- |
| Help | | |
| **Variáveis de entrada** | **Saída esperada** | **Classe de Eq. Cobertas** |
| Pressionar enter | Voltar à tela de menu inicial | V5 |
| Clicar sobre a tela | --- | I6 |

Tabela 2 - Particionamento por classes de equivalência para tela Help

|  |  |  |
| --- | --- | --- |
| Start | | |
| **Variáveis de entrada** | **Saída esperada** | **Classe de Eq. Cobertas** |
| Pressionar a tecla W, pressionar a tecla seta pra cima | Movimentos de bloco esquedo e direito pra cima | V6, V8 |
| Pressionar a tecla S, pressionar a tecla seta pra baixo | Movimentos de bloco esquedo e direito pra baixo | V7, V9 |
| Pressionar a tecla END | Velocidade volta a nível inicial do jogo | V10 |
| Pressionar a tecla ESC | Voltar à tela de menu inicial | V11 |
| Clicar sobre o bloco esquerdo | --- | I7, I9, I11, I13 |
| Pressionar teclas para esquerda e direita | --- | I8, I10, I12, I14 |

Tabela 3- Particionamento por classes de equivalência para tela Start

Com as tabela montadas, foi possível visualizar quais seriam as classes de equivalência válidas e inválidas, e com isso derivar casos de testes para cada uma delas.

A implementação dos casos de teste funcional foi feita, dentro do projeto base do jogo PongGame, no pacote test, na classe FunctionalTest.java.

Durante a construção dos testes, para simulação de um usuário final, utilizamos a classe java.awt.Robot.

Cada nome de método do teste unitário para testar funcionalidades possui quais classes de equivalência ele atende, para que, dessa forma, fique fácil saber qual o objetivo do método.

Prosseguindo com o desenvolvimento dos testes, aplicamos a técnica de grafo de causa-efeito para tentar identificar condições que as técnicas anteriores não contemplavam.

Foi identificado que, ao pressionar duas teclas juntamente, poderíamos ter um comportamento inesperado, como demonstrado no grafo abaixo

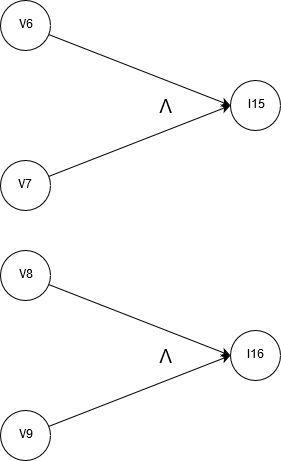


Figura 1- Grafo de causa-efeito para teclas pressionadas juntas no jogo

Com isso, nossa tabela de testes para o jogo (Start) foi atualizada com mais duas linhas ao final

|  |  |  |
| --- | --- | --- |
| Start | | |
| **Variáveis de entrada** | **Saída esperada** | **Classe de Eq. Cobertas** |
| Pressionar a tecla W, pressionar a tecla seta pra cima | Movimentos de bloco esquedo e direito pra cima | V6, V8 |
| Pressionar a tecla S, pressionar a tecla seta pra baixo | Movimentos de bloco esquedo e direito pra baixo | V7, V9 |
| Pressionar a tecla END | Velocidade volta a nível inicial do jogo | V10 |
| Pressionar a tecla ESC | Voltar à tela de menu inicial | V11 |
| Clicar sobre o bloco esquerdo | --- | I7, I9, I11, I13 |
| Pressionar teclas para esquerda e direita | --- | I8, I10, I12, I14 |
| Pressionar teclas W e S juntas | --- | I15 |
| Pressionar seta para cima e seta para baixo juntas | --- | I16 |

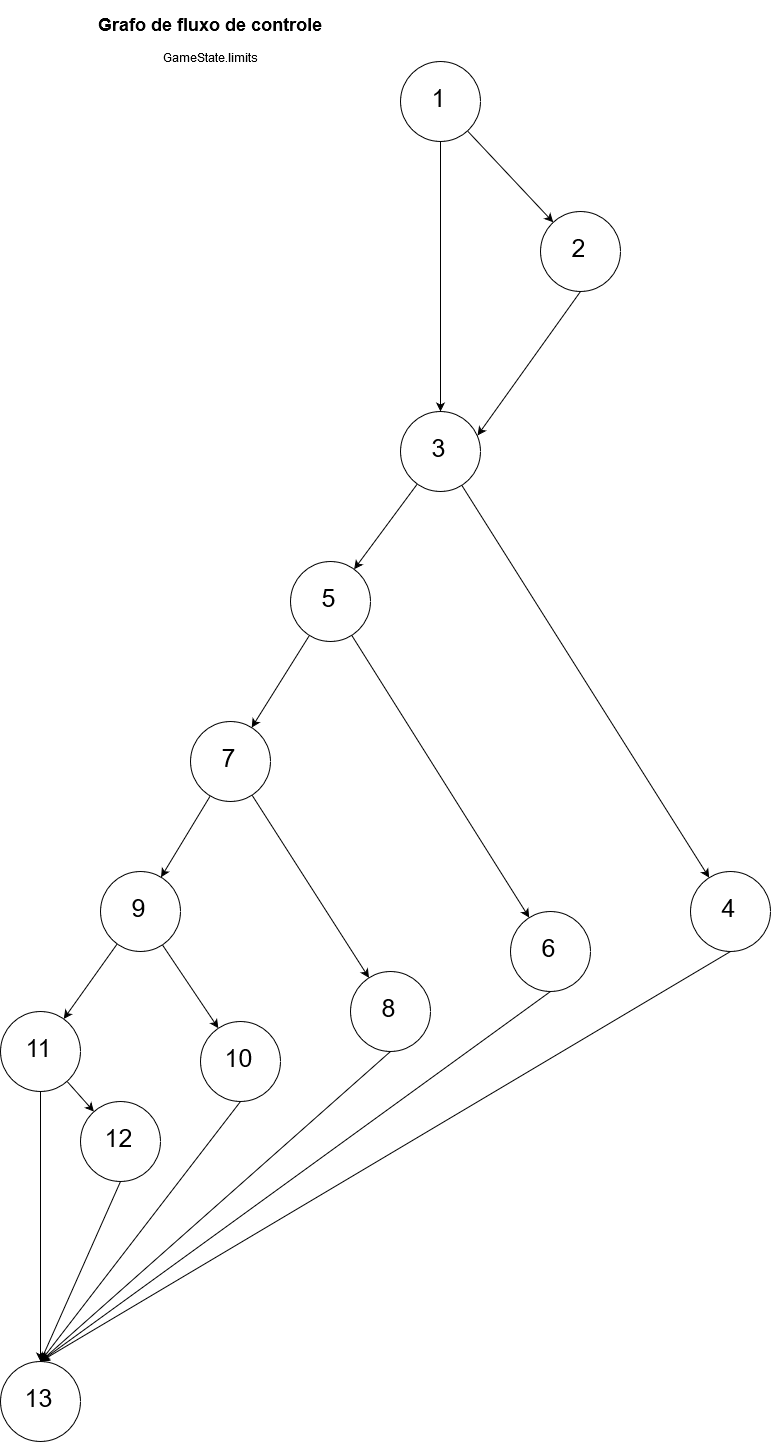
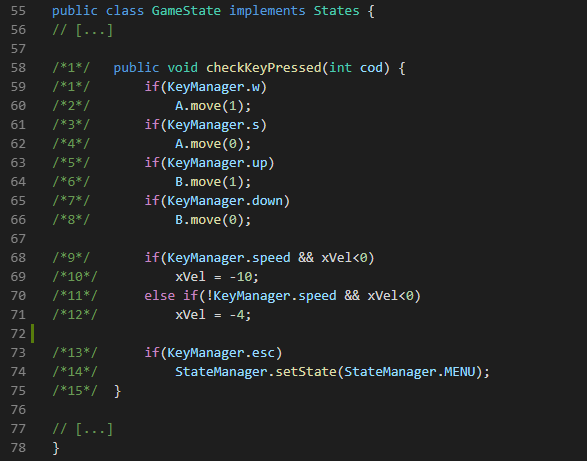
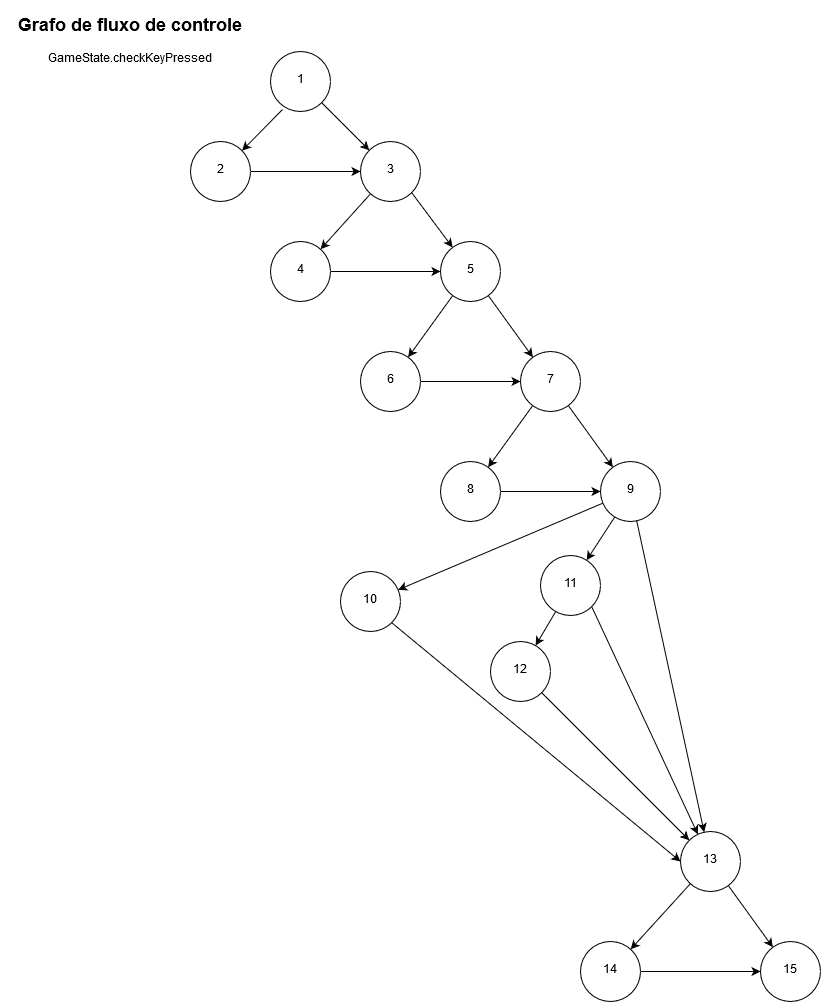
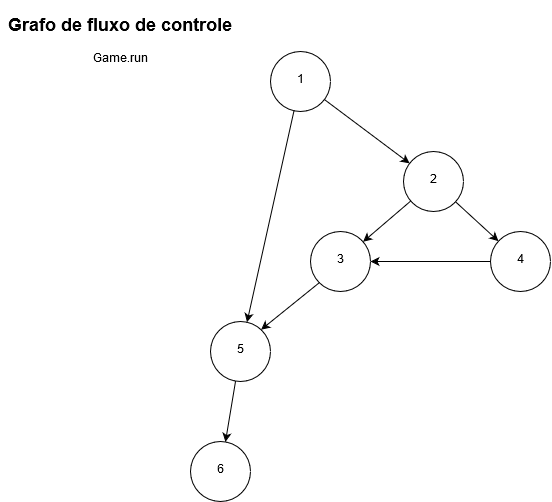
Tabela 4 - Tabela de particionamento de classes de equivalência após grafo de causa-efeito

Implementados os testes baseados nesses particionamentos, obtivemos uma cobertura de 95,20% com 11 métodos, métrica essa que foi obtida com a execução do EclEmma.

# 4 - Teste Estrutural

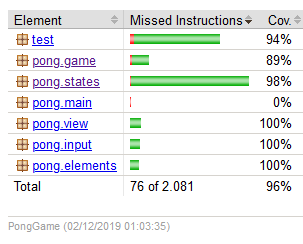
Para o teste estrutural, nos baseamos no relatório gerado pelo EclEmma para decidir em quais métodos do sistema atuar, com o propósito de aumentar a cobertura total do jogo.

Os GFCs (Grafos de fluxo de controle) foram gerados com base na análise do algoritmo de cada um dos métodos.

* GameState.limits()  
  
* GameState.checkKeyPressed()  
    
  
* Game.run()  
    
    
   

A implementação dos casos de teste estrutural foi feita, dentro do projeto base do jogo PongGame, no pacote test, na classe StructuralTest.java.

Com a implementação destes testes, a taxa de cobertura de testes, gerada pelo EclEmma chegou a 96,3%



# 5 - Teste de mutação

A ferramenta PIT Mutations entrava em loop infinito ao ser rodada para geração de mutantes. Em conversa com outros grupos que também utilizaram o PongGame como base para o trabalho, o ocorrido se repete.

# 6 - Referências

1 – PongGame - <https://github.com/Arthurk12/PongGame>

2 – Eclipse IDE - <https://www.eclipse.org/>

3 – Junit 5 - <https://junit.org/junit5/>

4 – PIT Mutations - <https://pitest.org/>