INFORMÁTICA NA EDUCAÇÃO: teoria & prática Porto Alegre, v.20, n.1, jan./abr. 2017. ISSN digital 1982-1654
ISSN impresso 1516-084X

JPlay Tutor: Uma nova abordagem para o ensino de programação utilizando jogos

JPlay Tutor: A new approach to programming teaching using games

ELANNE CRISTINA OLIVEIRA DOS SANTOS
Instituto Federal de Educação, Ciência e Tecnologia do Piauí

GLEISON BRITO BATISTA
Universidade Federal de Minas Gerais

ESTEBAN W. GONZALES CLUA
Universidade Federal de Fluminense

Resumo: O ensino de programação tem sido um desafio em universidades e escolas. Enquanto os ambientes de programação possuem sofisticadas ferramentas para detectar e indicar erros sintáticos, o mesmo não ocorre em relação a erros semânticos, podendo frustrar programadores aprendizes e inexperientes. Tendo em vista a proposta de auxiliar nos processos de ensino e aprendizagem foi utilizado o framework *JPlay*, que possui estruturas para o desenvolvimento de jogos simples em duas dimensões (*2D*). Este trabalho tem como propósito ampliar estudos anteriormente realizados, incluindo uma heurística baseada na análise estrutural de comportamentos de jogos desenvolvidos utilizando o *JPlay*. A heurística consiste em uma estratégia de comparação entre o programa do aluno e um programa modelo. Baseado nesta heurística, o sistema *JPlay* Tutor interpreta o código do aluno e retorna a análise detalhada. No artigo são apresentados uma breve descrição da heurística proposta e 2 estudos de caso elaborados com base nos resultados obtidos.
Palavras-chave: Ensino de programação. *JPlay*. Jogos. Heurística.

Abstract: The learning programming is being a challenge in universities and in schools. Whereas programming environments have sophisticated tools to detect and indicate syntax errors, the same does not occur with respect to semantic errors, which can frustrate learners and novice programmers. Based on the proposal of assisting the teaching and learning process, we propose the usage of *JPlay* framework that provides structures and architectures for developed with the approach of simple *2D* games. This work extends our previous one, including an heuristic based on the structural analysis of the behaviors of a *JPlay* program. The heuristic consists on a comparison between the student program and a model program. Based on this heuristic, the *JPlay Tutor* system interprets the student code and return detailed analysis. In this paper we present a brief review on the heurist employed and 2 case studies based on the achieved results.
Keywords: Programming teaching. *Jplay*. Games. Heuristic.

DOS SANTOS, Elanne Cristina Oliveira; BATISTA, Gleison Brito; CLUA, Esteban W. Gonzales. JPlay Tutor: Uma nova abordagem para o ensino de programação utilizando jogos. *Informática na Educação*: teoria & prática, Porto Alegre, v. 20, n.1, p. 81-102, jan./abr. 2017.

1 Introdução

O ensino de algoritmos e programação consiste em um grande desafio, não somente em universidades, mas também em escolas e centros de treinamento. Há diversos estudos que demonstram a dificuldade encontrada nos processos de ensino e de aprendizagem nas disciplinas relacionadas a programação, resultando em uma larga evasão nos cursos de computação (PINHEIRO et al., 2007)(BARBOSA et al., 2011). Aponta-se como uma das principais causas para tal dificuldade, por parte dos alunos, a de entender os conceitos abstratos envolvidos no processo de programação de computadores (SANTOS e RAPKIEWICZ, 2007). Recentemente têm surgido várias propostas voltadas a estimular este processo (ALLEN et al., 2002) (KOLLING et al., 2003) (TRAETTEBERG e AALBERG, 2006) (ALLOWATT e EDWARDS, 2005).

Também objetivando auxiliar na tarefa de ensino de programação, Feijó et al. (2010) propõem o uso do *framework JavaPlay*. Por meio desse *framework*, é possível utilizar a lógica de desenvolvimento de jogos simples no ensino dos conceitos envolvidos na atividade de programação. Essa ferramenta fornece aos estudantes uma maneira fácil de desenhar e movimentar imagens na tela do computador, além de fornecer as funcionalidades necessárias para o desenvolvimento de jogos *2D* utilizando a linguagem de programação *Java*. Em seguida, alguns dos autores deste trabalho propuseram uma remodelagem do JavaPlay e criaram o *JPlay* (JPLAY,2012).

Em estudos anteriores, foi proposto um analisador semântico baseado na comparação entre o comportamento de dois programas que utilizam o *framework JPlay*: programa modelo e programa do estudante. Os resultados estão baseados no algoritmo de comparação entre os pares de classes similares (par <classe do programa modelo, classe do programa do estudante>) (SANTOS et al., 2014a).

Em um trabalho seguinte foi apresentada a heurística baseada na análise estrutural de comportamentos e os seus quatro níveis de desenvolvimento (SANTOS et al., 2014b). Neste artigo será apresentada uma breve descrição da heurística adotada, além de dois estudos de caso baseados nos resultados obtidos por meio da utilização da ferramenta *JPlay Tutor*.

2 Trabalhos Relacionados

Diferentes abordagens podem ser utilizadas no processo de ensino-aprendizagem de programação. Tais abordagens são desenvolvidas com o objetivo de melhorar e aumentar o engajamento dos alunos durante o processo de aprendizagem. Baseado nas classificações de trabalhos anteriores (DELGADO, 2005) (PINHEIRO et al., 2007), definiu-se uma classificação possível das técnicas relacionadas a aprendizagem de programação como: programação baseada em teste unitário, propostas de ambiente de programação, avaliador automático, análise de padrões de programação e sistemas de depuração automática (sistemas de tutoria inteligente e sistemas de diagnóstico de programas).

Na programação baseada em testes unitários o professor elabora um conjunto de testes específicos para resolver um problema particular e o estudante deve construir um programa que permite obter os resultados esperados durante a execução de todos os testes (PINHEIRO et al., 2007) (TRAETTEBERG e AALBERG, 2006).

A proposta de ambientes de programação consiste no fato de que algumas ferramentas de desenvolvimento foram criadas com o objetivo de ajudar estudantes a aprender a programar, tais como *BlueJ* (KOLLING et al.,2003) e *DrJava* (ALLEN et al.,2002). Segundo Kolling et al. (2003) a principal vantagem do ambiente *BlueJ* é a simplicidade, considerando que um dos grandes problemas de muitos ambientes de programação é a complexidade.

O avaliador automático é usado para ajudar os professores com tarefas de correção de atividades. O professor pode definir testes para serem executados automaticamente depois que os estudantes enviarem suas atividades de programação e os resultados dos testes podem ser

usados para definir a nota do estudante (PINHEIRO et al., 2007). O *Web-Cat* (ALLOWATT e EDWARDS, 2005) é um exemplo de ferramenta que utiliza essa metodologia.

É possível citar também o sistema Run.Codes (RUN.CODES,2016). Os alunos podem submeter uma atividade por meio de *upload* no sistema. Uma limitação consiste no fato de que as análises só suportam bibliotecas nativas (pacotes importados Java, por exemplo, não são reconhecidos). O Run.Codes também apresenta uma opção para inclusão de casos de testes.

A análise de padrões de programação é baseada na pesquisa de sugestões de aprendizagem anteriores, devido à crença de que os programadores experientes procuram resolver problemas baseados em soluções anteriores, relacionadas com o problema novo, que podem ser adaptadas para a situação ideal (DELGADO, 2005). Experiências anteriores são a base para padrões de programação (ALEXIS e DELLER, 2013). Os sistemas Proust, aplicado ao ensino da linguagem de programação Pascal, proposto por (JOHNSON e SOLOWAY, 1984) e PROPAT (DELGADO, 2005) são sistemas que utilizam tal estratégia. Tais soluções têm como limitação o fato de que para a obtenção de um bom diagnóstico do programa é necessária uma ampla biblioteca de planos (conjunto de objetivos utilizados para solucionar o problema), uma vez que deve haver planos correspondentes a várias maneiras que os estudantes resolvem problemas.

Um sistema de depuração automática é um sistema que utiliza técnicas com o objetivo de encontrar e classificar os componentes de um programa. Com base no tipo de técnica utilizada, ele pode ser classificado em sistemas de tutores inteligentes de programação e sistema de diagnóstico de programa (DELGADO, 2005). LAURA é uma das primeiras tentativas de construir um sistema de tutoria para o ensino de programação. A ferramenta foi desenvolvida na Universidade de Caen, na França, e é um sistema de diagnóstico de programa escrito em Fortran (BOTELHO, 2010). Em sua estratégia LAURA se utiliza de uma solução de referência (um programa) para diagnosticar o programa do aluno, ou seja, o sistema realiza a comparação entre dois programas, o modelo e o candidato. A comparação é possível por meio da representação do programa modelo e do programa candidato por meio de grafos, e a sua estratégia heurística é identificar passo a passo os elementos semelhantes dos grafos (ADAM e LAURENT, 1980). Santos et al. (2014a) também afirma que LAURA foi o único em seu ramo de atuação a utilizar esta estratégia.

Também classificado como uma ferramenta usada para diagnóstico automático de programas, podemos citar o sistema apresentado neste artigo, denominado de Tutor JPlay, que tem o objetivo de analisar semanticamente jogos 2D (construídos com Java e o *framework JPlay*) construídos por estudantes. O sistema é capaz de realizar um diagnóstico com objetivos educacionais, tendo a função de interpretar semanticamente e arquiteturalmente um programa Java desenvolvido usando o JPlay e retornar resultados desta análise para o aluno (SANTOS et al., 2014a). O trabalho apresentado difere-se dos demais por ser baseado em padrões de projetos simples utilizados na construção de jogos, seguindo o propósito do *framework JPlay* (FEIJÓ et al., 2010). Outro diferencial consiste na comparação entre variáveis que contêm comportamento similar. Na ferramenta LAURA, por exemplo, são gerados dois grafos do programa completo (um para o modelo e outro para o candidato), que em seguida são usados na comparação. Na abordagem proposta no presente trabalho é apresentada uma estrutura de dados denominada árvore de comportamentos para cada variável dos programas. Para evitar problemas de diferenças encontradas devido a variações de programação (várias maneiras de resolver o mesmo problema em um programa) foi definida a estratégia de padronização do código-fonte, obtendo assim um maior nível de granularidade na análise dos comportamentos dos jogos desenvolvidos pelos alunos.

3 *JPlay* Tutor e a heurística baseada na comparação dos comportamentos dos programas

O algoritmo utilizado na ferramenta *Jplay* Tutor é uma heurística que consiste na comparação de dois programas, o programa do aluno e o programa modelo (escrito pelo

professor). O objetivo é fazer uma análise do código desenvolvido pelo estudante com o objetivo de guiá-lo durante o desenvolvimento de um jogo específico.

Inicialmente o aluno seleciona o programa modelo com o qual ele vai comparar o seu programa, conforme mostra a Figura 1.

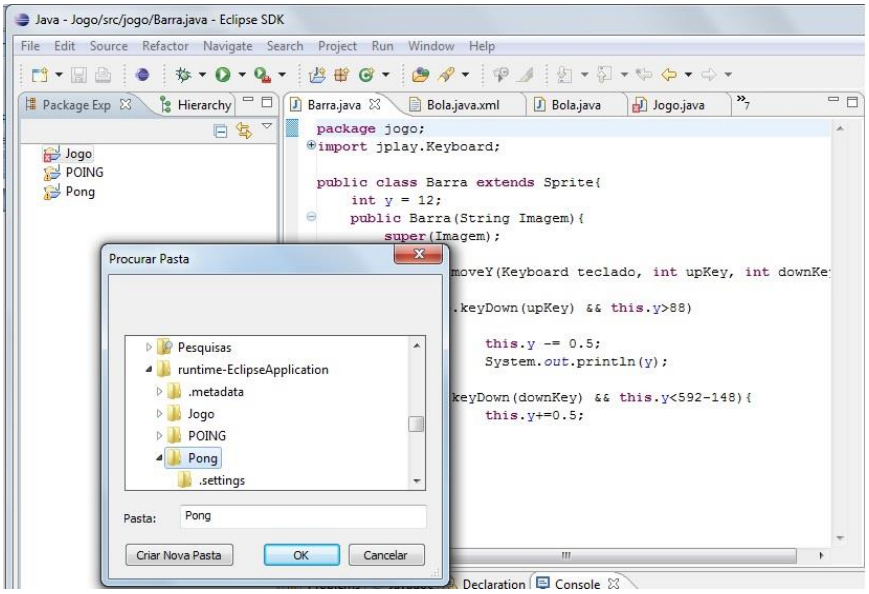


Figura 1 - Escolhendo o programa modelo no plugin *JPlay* Tutor

A primeira fase da comparação classifica pares de classes similares. Para classificar os pares de classes são utilizadas regras específicas que estão descritas em Santos et al. (2014a). Após a classificação dos pares de classes similares, o *JPlay* Tutor classifica os pares de variáveis similares para cada par de classes. Este artigo se concentra na análise de resultados da comparação entre os pares de classes formados. A comparação é composta por duas etapas: padronização do modelo e classificação dos pares de variáveis.

3.1 Padronização do Modelo

Com o propósito de padronizar o código desenvolvido pelo estudante de acordo com o modelo elaborado pelo professor, foi adotada a estratégia de que sejam inclusos marcadores específicos para cada classe do programa modelo. Os marcadores devem ser declarados no início de cada classe. Existe um comentário associado a cada marcador. Cada comentário contém uma especificação relativa ao comportamento que classe deve seguir, tais como, a quantidade de métodos que ela deve conter, se ela herda ou não as características de outra classe, qual seria esta outra classe etc. Abaixo alguns marcadores:

- *Inheritance*: identifica a super-classe, caso tenha sido utilizada *herança*;
- *Constructor*: Informa a declaração de um construtor na classe;
- *Keyboard*: Identifica a utilização de um objeto do tipo *Keyboard* na classe, ou seja, a utilização do teclado.
- *Mouse:* Identifica a utilização de um objeto do tipo *Mouse* na classe, ou seja, a utilização do *mouse*.

3.1 Classificação dos Pares de Variáveis

Com o propósito de comparar pares similares de variáveis formados pelo analisador, é proposta uma estrutura denominada árvore de comportamentos. Essa estrutura contém todos os comportamentos de uma determinada variável. Foram definidos três tipos de comportamentos:

- *Assignment*: Identifica atribuição de valores à variável. No exemplo, o trecho de código mostra um comando de atribuição associado à variável *"cont"* : *cont := 0;*
- *Conditional: Identifica a utilização de uma estrutura condicional. No exemplo, o trecho de código mostra um comando condicional associado à variável "cont": if (cont > -1).*
- *Loop: Identifica a utilização de estruturas de repetição. No exemplo, o trecho de código mostra um comando de repetição associado à variável "cont": while (cont>-1).*

Para cada variável dos pares similares, é gerada uma árvore de comportamento. Essas árvores são comparadas, de forma a identificar diferenças entre os comportamentos das variáveis. Caso ocorram diferenças nas árvores, é possível que o programa do estudante não esteja realizando o comportamento estipulado no programa modelo. Como exemplo, a Figura 2 mostra a comparação entre as variáveis *"left"* e *"carry2"*. Pode-se perceber que a variável *"carry2"* contém um comportamento *Assignment* a mais que a variável *"left"*. Nesse caso a mensagem associada ao comportamento em que foi observada a diferença será enviada para o aluno. A mensagem associada a este comportamento pode ser visualizada na Figura 3.

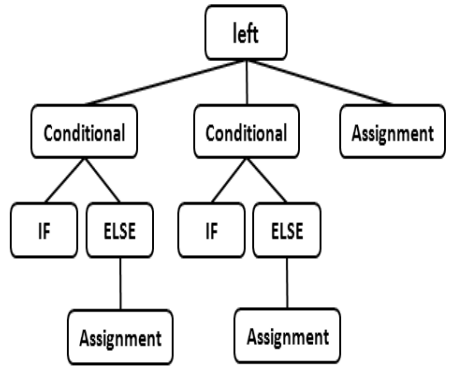
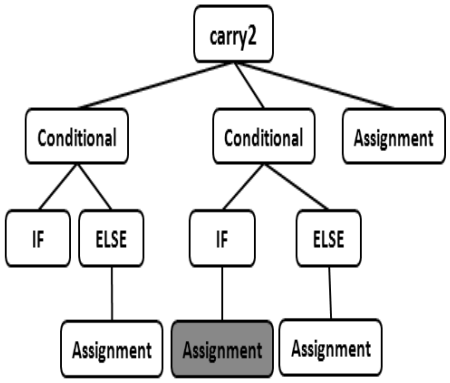
(a) Árvore de comportamento da variável *"left"*	(b) Árvore de comportamento da variável *"carry2"*
	

Figura 2 - Exemplo de comparação entre as variáveis de comportamento das variáveis *"left"* e *"carry2"*

```
public Ball(String Imagem){
      super(Imagem);
     /*
      Inicialize o atributo que controla o movimento inicial da bola para
      a esquerda ou para a direita no construtor da classe
     */
    left=true;
}
```

Figura 3 - Exemplo de comportamento *"assignment"* da variável *"left"* da classe *"Ball"* do programa modelo

Ao encontrar diferença(s) de comportamento entre os programas, o *JPlay* Tutor pergunta se ele quer visualizar as árvores de comportamento, conforme mostra o exemplo da Figura 4.

Caso o usuário responda afirmativamente, o analisador constrói as árvores de comportamento do par de variáveis, sendo possível visualizar, por meio da comparação das árvores, a(s) diferença(s) encontradas, conforme mostra o exemplo da Figura 5.

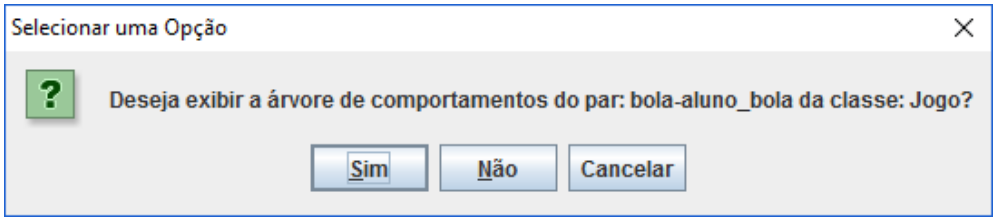


Figura 4 - *JPlay* Tutor pergunta ao usuário se deve mostrar as árvores de comportamento do par de variáveis *"bola"* e *"aluno_bola"*

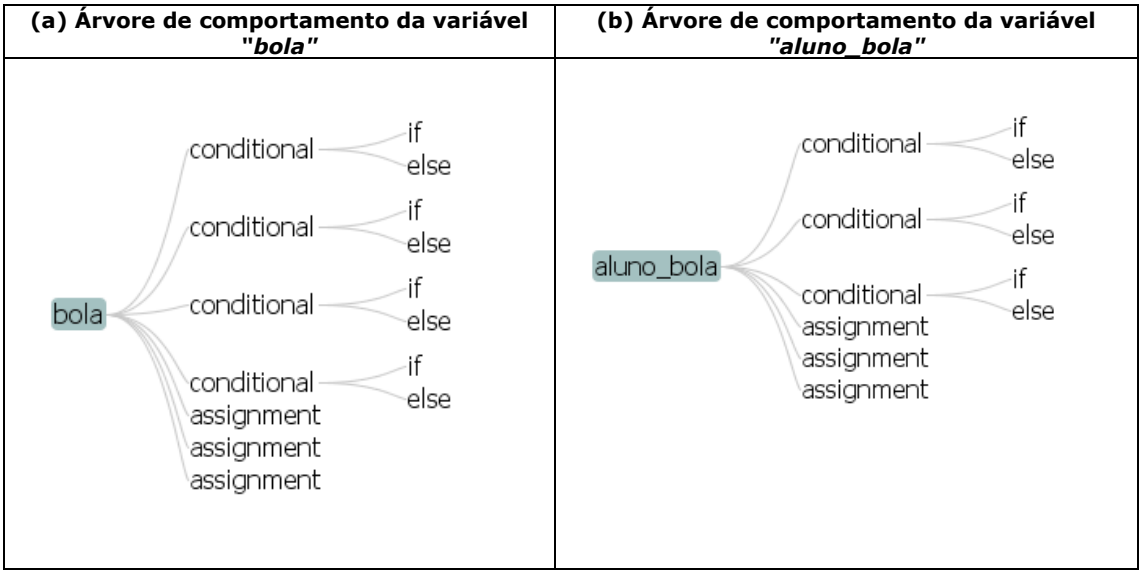


Figura 5 - Exemplo de comparação entre as variáveis de comportamento das variáveis *"bola"* e *"aluno_bola"*

4 Metodologia do Estudo de Caso

Baseando-se em uma abordagem metodológica do tipo qualitativa, a análise desenvolvida neste trabalho é caracterizada como um estudo de caso e tem o objetivo de executar a validação do sistema analisador semântico.

São realizados 2 (dois) estudos de caso. Procura-se com esta escolha validar dois tipos de programas desenvolvidos usando *Java* e o *framework JPlay*:

- O primeiro programa foi aplicado em uma prova, e se caracteriza por ser um desafio mais simples (não é um jogo) no sentido de que as funcionalidades necessárias a serem implementadas estão divididas em 2 (duas) questões de uma prova e elas podem ser desenvolvidas em uma única classe (classe main) do projeto Java. Nesse primeiro experimento avaliaram-se 8 (oito) provas.
- O segundo programa proposto foi o jogo BrickBreak, sendo o mesmo aplicado durante o minicurso Desenvolvimento de jogos usando Java e framework JPlay. Participaram do minicurso alunos dos cursos Técnico em Informática integrado ao Ensino Médio, Técnico Subsequente em Informática e Tecnologia em Análise e Desenvolvimento de Sistemas. O jogo envolve a necessidade de mais conhecimentos sobre o uso do paradigma de orientação a objetos, visto que é proposta a divisão do projeto em cinco classes (Barra, Bloco, Bola, Jogo e Principal), e em cada uma das classes o jogo envolve a

implementação de funcionalidades específicas. Um total de 10 (dez) programas foram analisados nesse experimento.

Desta forma procura-se generalizar os resultados obtidos a partir do analisador semântico aplicado a programas básicos usando *Java* e o *framework JPlay*. Um programa básico caracteriza-se, neste caso, por envolver poucas funcionalidades (o programa não é necessariamente um jogo), sendo desenvolvido somente com o uso da classe principal *main* (caso do primeiro programa) e para programas mais complexos (programas que usam mais que uma classe no projeto e necessitam de mais conhecimento sobre o paradigma de orientação a objetos, como no caso do segundo programa).

A validação de um sistema consiste no processo de julgar quão bem um sistema resolve o problema para o qual ele foi concebido. Considerando que o algoritmo utilizado no analisador semântico consiste em uma heurística, ou seja, baseia-se em aproximações de um resultado exato, o algoritmo vai procurar encontrar as melhores soluções possíveis, mas não é possível garantir, em todas as ocorrências de sua aplicação, soluções ótimas ou exatas (WAINER, 2007).

Assim, por meio dos estudos de caso procuramos responder à seguinte questão: O analisador é eficaz em apontar problemas oriundos das diferenças entre a comparação de dois programas (o programa do aluno e o programa modelo do professor)?

Para responder essa questão foram realizadas duas análises:

- Análise quanto à padronização: é estabelecida por meio de configuração de marcadores, definidos pelo professor no início de cada classe do programa.
- Análise quanto à detecção de diferenças de comportamentos entre as variáveis dos programas: a detecção de diferença de comportamentos pode gerar 2 (dois) tipos de resultados, que consistem em diferenças nos comportamentos dos pares formados pelo analisador (formação das árvores de comportamento) e comportamentos das variáveis que não conseguiram formar pares, sendo que isto acontece no caso do analisador não encontrar nenhuma variável no programa do aluno que seja do mesmo tipo de uma variável que se encontra no programa do professor.

Os resultados da análise são apresentados em duas etapas: (i) uma análise geral, em que são verificados os dados quantitativos dos resultados e (ii) uma análise mais detalhada, em que é realizado um estudo mais específico sobre os comentários sugeridos pelo analisador.

4.1 Matriz de Confusão

A matriz de confusão é uma tabela específica para avaliar o desempenho de um algoritmo ou método de classificação. As colunas da matriz representam classes previstas, enquanto as linhas representam as classes reais dos elementos analisados (HASTIE et al., 2009).

As células da matriz de confusão que correspondem ao cruzamento das informações das classes previstas e das classes reais representam a quantidade de positivo-positivos (*PP*), negativo-negativos (*NN*), falso-negativos (*FN*) e falso-positivos (*FP*).

A matriz de confusão deste trabalho é construída a partir da observação dos comentários emitidos pelo analisador sobre uma determinada classe de programa. Os comentários são sugestões encontradas no programa, são associados aos comportamentos e inseridos pelo próprio professor no programa modelo. Define-se:

- *PP*: Quando a classe de programa não apresenta comentários e o comportamento da classe está realmente correto.
- *NN*: Quando um comentário detecta um erro de comportamento da classe e realmente existe um problema no comportamento.
- *FP*: Quando a classe de programa não apresenta comentários, mas o comportamento da classe está incorreto.
- *FN*: Quando um comentário detecta um erro de comportamento da classe, no entanto não existe erro no comportamento.

Um programa dado como *correto* não apresenta diferenças na comparação com o programa modelo, logo não gera nenhum comentário ao aluno. Este resultado caracteriza-se como *PP* (positivo-positivo) ou *FP* (falso-positivo).

Um programa dado como do tipo *incorreto* apresenta diferenças na comparação com o programa modelo, logo gera comentários resultantes dessas diferenças. Este resultado pode ser classificado como *NN* (negativo-negativo) ou *FN* (falso-negativo).

A classificação de uma classe do tipo positivo-positivo vs. falso-positivo, negativo-negativo vs. falso-negativo será realizada por meio da observação do pesquisador (o próprio professor da disciplina/minicurso) baseado no real funcionamento do programa.

4.2 Sensibilidade, Especificidade, Acurácia e Precisão

Os resultados obtidos a partir do diagnóstico do analisador são verificados através da análise dos comentários emitidos por ele. Assim, através da matriz de confusão, foram classificados os tipos de comentários como falso-negativo e negativo-negativo. Considerando o tipo e quantidade de cada um dos comentários, é possível validar o diagnóstico do analisador para um programa ou classe,

Sendo possível verificar os valores da sensibilidade, especificidade, eficiência, acurácia e precisão, baseado na classificação dos comentários impressos. Essa validação é utilizada para aferir o poder discriminativo do *JPlay Tutor* como bom ou não para uma determinada análise.

Sensibilidade e especificidade são medidas estatísticas de desempenho de técnicas de classificação calculadas utilizando informações presentes na matriz de confusão. Associando estas definições ao diagnóstico apresentado pelo *JPlay Tutor* para uma determinada classe de programa, tem-se a sensibilidade como a capacidade de classificar como correta uma classe de programa realmente correta, e a especificidade como a capacidade de classificar como incorreta uma classe de programa realmente incorreta.

A acurácia refere-se ao grau de conformidade e correção de algo quando comparado com um valor verdadeiro, enquanto a precisão refere-se a um estado de exatidão estrita. Assim, a precisão de um experimento é uma medida da confiabilidade e consistência. A acurácia de um experimento é uma medida de quão próximos os resultados estão em relação ao valor aceito (DIFFEN, 2015). Neste artigo, a acurácia significa o quanto o *JPlay Tutor* se aproximou dos resultados reais. A precisão significa o quanto o *JPlay Tutor* identificou resultados reais. Podem ser calculados dois índices de precisão, sendo estes índices conhecidos como preditividade positiva e preditividade negativa. A preditividade positiva indica a proporção de resultados positivos que realmente são positivos. A preditividade negativa indica a proporção de resultados negativos que realmente são negativos (WIKIHOW, 2015). A Tabela 1 descreve as medidas.

Tabela 1 – Medidas para aferir a qualidade do analisador JPlay Tutor baseadas no valores PP (positivo-positivo), FN (falso-positivo), NN (negativo-negativo) e FN (falso-negativo)

	Fórmula
Sensibilidade	$Sensibilidade = \frac{PP}{PP + FN}$
Especificidade	$Especificidade = \frac{NN}{NN+FP}$

Acurácia	$Acurácia = \frac{PP + NN}{PP+FP+NN+FN}$
Preditividade Positiva	$Preditividade\ Positiva = \frac{PP}{PP+FP}$
Preditividade Negativa	$Preditividade\ Negativa = \frac{NN}{NN+FN}$

4.3 As Três Hipóteses Propostas

Quando são detectadas diferenças entre a comparação de programas e comentários são exibidos como sugestões ao aluno, para medir a precisão do *JPlay Tutor* verifica-se os seguintes percentuais:

- Percentual de negativo-negativo (*PNN*): total de comentários negativo-negativo/total de comentários.
- Percentual de falso-negativo (*PFN*): total de comentários falso-negativo/total de comentários.

Baseado nesses valores, uma avaliação eficaz do analisador indicará alguma das seguintes hipóteses:

Hipótese 1: um percentual de negativo-negativo sempre maior que um percentual de falso-negativo (PNN > PFN).

Para os casos positivo-positivo e negativo-positivo, quando diferenças não são detectadas pelo *JPlay Tutor* e, portanto, não existem comentários, é considerado, como resultado, a observação do pesquisador (o funcionamento/execução do programa pode ser considerado pelo observador como correto ou incorreto). Assim realiza-se os seguintes percentuais:

- Percentual de positivo-positivo (*PPP*): total de programas positivo-positivo/total de programas de comportamento correto.
- Percentual de falso-positivo (*PFP*): total de programas falso-positivo/total de programas de comportamento correto.

Baseado nesses valores uma avaliação eficaz do analisador indicará a seguinte hipótese, para o caso de valores PPP e PFP diferentes de 0 (zero):

Hipótese 2: um percentual de positivo-positivo sempre maior que um percentual de falso-positivo (PPP > PFP).

Para os casos em que não existirem ocorrências de positivo-positivo (PPP=0), uma avaliação eficaz do analisador indicará a seguinte hipótese:

Hipótese 3: percentual positivo-positivo igual a percentual falso-positivo (PPP=PFP).

5 Estudo de Caso 1: Prova Aplicada na Disciplina *Tópicos Especiais de Desenvolvimento de Software*

O primeiro estudo de caso se concentrou em códigos de programas que eram formados por uma única classe (todos os códigos continham somente a classe *main* do projeto). O programa foi aplicado como prova prática para os estudantes da disciplina de *Tópicos Especiais em Desenvolvimento de Software* do curso Técnico em Informática integrado ao Ensino Médio, sendo a disciplina oferecida no 4o. ano do curso e tendo participado 8 alunos do experimento. O enunciado da prova, foi dividido em 2 (duas) questões. A primeira questão foi dividida em 2 (dois) itens:

Questão 1.1: Quando o usuário pressionar o botão esquerdo do *mouse*, um círculo deve ser desenhado na área da janela apontada pelo *mouse* e adicionada na lista de círculos, conforme mostra a Figura 6.

Questão 1.2: Quando o usuário pressionar a tecla *ESC*, o programa deve ser encerrado.

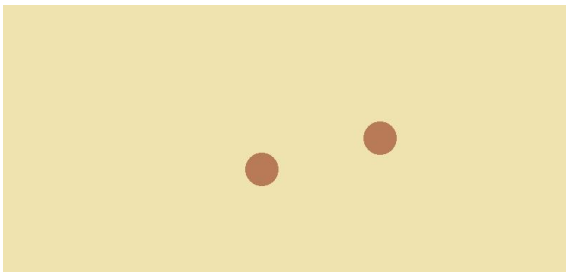


Figura 6 – Quando o usuário pressionar o botão esquerdo do mouse, os círculos devem ser desenhados na janela do jogo e adicionados à lista de círculos.

Na segunda questão do teste o desafio foi verificar se a área onde o usuário pressionou o botão esquerdo do *mouse* já está ocupada por um círculo ou não. Se a área já estiver ocupada, a mensagem com a informação *Área já ocupada!* deve aparecer. Se a área não estiver ocupada, o novo círculo deve ser adicionado na lista de círculos e desenhado na janela do jogo na posição corrente do *mouse*.

5.1 Resultados Obtidos por meio da Medição de Sensibilidade, Especificidade, Acurácia e Precisão

Foram verificados dois tipos de resultados: o primeiro baseado por unidade de programa analisado, assim são verificados todos os comentários que ocorreram durante a análise e seus respectivos tipos (Negativo-Negativo, Falso-Negativo) ou, no caso do programa não apresentar comentários, é verificada a análise do observador sobre o programa (Falso-Positivo, Positivo-Positivo). Então o programa é classificado de acordo com o tipo que possui o maior peso de classificação. O maior peso de classificação é de um comentário do tipo Negativo-Negativo, seguido pelos Falso-Negativo, Falso-Positivo e Positivo-Positivo.

Assim, mesmo que o programa possua vários comentários de tipos diferentes, ele será classificado com o tipo de maior peso. O objetivo é considerar que, se o programa apresenta pelo menos um comentário incorreto, pode-se concluir que todo o programa será classificado como de um tipo incorreto.

O segundo tipo de resultado é baseado na quantidade de comentários apresentados por cada programa, em que a classificação do tipo de cada um dos comentários é quantificada. Neste caso, é importante observar que, para calcular os resultados da análise, serão consideradas as quantidades de comentários Negativo-Negativo e Falso-Negativo, mas como os

resultados do tipo Positivo-Positivo e Falso-Positivo não resultam em comentários, nestes dois casos, serão considerados uma ocorrência do tipo Positivo-Positivo ou uma ocorrência do tipo Falso-Positivo.

As Tabelas 2 e 3 apresentam os resultados das medidas de eficiência obtidas por meio dos dois resultados.

Tabela 2 - Medida de Eficiência do Sistema Analisador de Acordo com o total de programas analisados para o Primeiro Estudo de Caso, para uma amostragem de 8 alunos.

	Medidas de eficácia do sistema analisador JPlay Tutor de acordo com o total de programas analisados		
Classe "main" do projeto	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*2*	*8*
	FN	*0*	*0*
	PP	*6*	*0*
	FP	*0*	*0*
	Sensibilidade	*100%*	*-*
	Especificidade	*100%*	*100%*
	Acurácia	*100%*	*100%*
	Preditividade Positiva	*100%*	*-*
	Preditividade Negativa	*100%*	*100%*

Tabela 3- Medida de Eficiência do Sistema Analisador de Acordo com o total de comentários analisados para o Primeiro Estudo de Caso, para uma amostragem de 8 alunos.

	Medidas de eficácia do sistema analisador de acordo com o total de comentários analisados		
Classe "main" do projeto	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*2*	*29*
	FN	*0*	*9*
	PP	*6*	*0*
	FP	*0*	*0*
	Sensibilidade	*100%*	*0%*
	Especificidade	*100%*	*100%*
	Acurácia	*100%*	*76%*
	Preditividade Positiva	*100%*	*-*
	Preditividade Negativa	*100%*	*76%*

6 Estudo de Caso 2: Jogo BrickBreak

O jogo *BrickBreak* foi desenvolvido como atividade do minicurso *Desenvolvimento de Jogos usando Java e o framework JPlay*. 10 alunos desenvolveram o jogo durante dois dias de aula. Inicialmente os comportamentos de cada uma das classes envolvidas no projeto foram apresentados aos alunos. O enunciado divide as orientações por classe de projeto, de forma que para cada classe cada um dos comportamentos foi discutido em sala de aula, bem como as maneiras de definir aquele comportamento em termos de implementação (quais métodos Java

e *JPlay* estariam envolvidos, por exemplo). As classes *Bola, Barra* e *Bloco* foram as primeiras a serem desenvolvidas e as classes *Jogo* e *Principal* foram as últimas classes desenvolvidas.

As classes *Bola*, *Barra* e *Bloco* definem os respectivos comportamentos dos personagens (objetos) do jogo: *bola*, *barra* e *bloco*. A Figura 7 mostra os objetos do jogo.

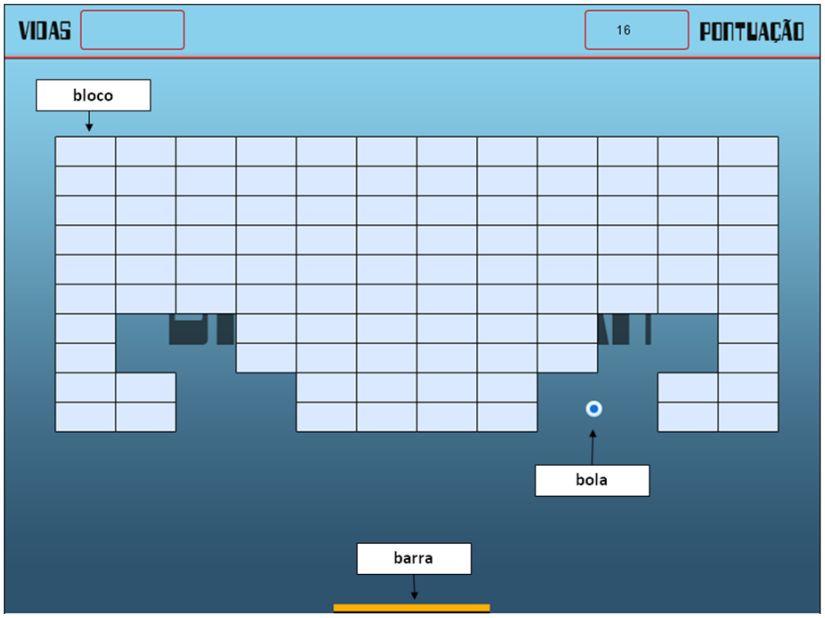


Figura 7 – Janela do jogo *BrickBreak*, com a respectiva indicação dos personagens do jogo.

De acordo com o enunciado apresentado aos alunos, a classe *Bola* possui duas especificações básicas: a bola deve se mover no eixo x e y. Assim, na definição do modelo do professor, a classe *Bola* possui três métodos, sendo eles o construtor da classe, um método para realizar o movimento no eixo *x* e um método para realizar o movimento no eixo *y*. A Figura 8 apresenta os marcadores de padronização da classe.

```
/**
*
* @comment A classe Bola possui 2 especificações básicas: deve se mover no
eixo x e y.
* @method 3
* @ comment A classe Bola deve possuir 3 métodos:o construtor da classe, um
método para realizar o movimento no eixo x e um método para realizar o
movimento no eixo y.
* @inheritance Sprite
* @comment A classe deve herdar de Sprite
* @ movex true
* @comment realize o movimento no eixo do x
* @ movey true
* @comment realize o movimento no eixo do y
*/
public class Bola extends Sprite{
..............
```

Figura 8 - Marcadores de padronização da classe *Bola.*

De acordo com o enunciado apresentado aos alunos, a classe *Barra* também possui 2 (duas) especificações: a barra deve se mover para os lados esquerdo e direito utilizando o controle do teclado. Assim, na definição do modelo do professor, a classe *Barra* possui dois métodos: o

construtor da classe e um método para realizar o movimento para os lados esquerdo e direito por meio do teclado. A Figura 9 apresenta os marcadores de padronização da classe.

De acordo com o enunciado apresentado aos alunos, a classe *Bloco* não possui especificações, considerando que o objeto bloco permanece estático durante o decorrer de todo o jogo e não possui ação específica. Apenas a colisão da bola com este objeto, faz com que ele seja destruído, mas esta ação de colisão é tratada pela classe *Jogo*. Portanto, na definição do modelo do professor, a classe *Bloco* possui somente o método construtor. Por este motivo esta classe não foi inserida na análise de resultados. A Figura 10 apresenta os marcadores de padronização da classe *Bloco.*

```
/**
*
* @comment A classe deve herdar de Sprite e possuir 2 métodos: o construtor
que deve inicializar a figura da barra, e um método para controlar o
movimento no eixo x usando o teclado (DICA: use o método keydown() da classe
Keyboard do JPlay)
* method 2
* @comment A classe deve conter 2 métodos: o construtor que deve inicializar
a figura da barra, e um método para controlar o movimento no x usando o
teclado (DICA: use o método keydown() da classe Keyboard do JPlay)
* @inheritance Sprite
* @comment A classe Barra deve herdar de Sprite
* @movex true
* @comment A classe deve realizar o movimento no eixo x para a esquerda e
para a direita
* @keyboard true
* @comment O movimento da classe Barra no eixo x deve ser controlado pelo
teclado (DICA: use o método keydown() da classe Keyboard do JPlay)
*/
public class Barra extends Sprite{
.............
```

Figura 9 -Marcadores de padronização da classe *Barra*

```
**
/* @comment O objeto bloco não possui nenhuma ação específica e permanece
estático durante o decorrer de todo o jogo. Apenas a colisão da bola com
este objeto faz com que ele seja destruído, mas esta ação de colisão é
tratada pela classe "Jogo".
* @method 1
* @comment A classe Bloco possui somente o método construtor da classe
* @inheritance Sprite
* @comment A classe Bloco herda de Sprite
*/
public class Bloco extends Sprite {
..................
```

Figura 10 -Marcadores de padronização da classe *Bloco*

A classe *Jogo* define o comportamento do *gameloop* do jogo. Ela é a classe que possui a maior quantidade de comportamentos a serem implementados, pois o *gameloop* define o padrão de projeto central do jogo, ou seja, nesta classe os objetos devem ser instanciados e interagir entre si de acordo com as regras estabelecidas pelo jogo (as colisões entre os objetos bola e barra, bola e bloco são tratadas nesta classe). A Figura 11 apresenta os marcadores de padronização da classe *Jogo.*

```
/**
* @comment Esta classe deve conter o loop infinito do jogo (coração da
execução do jogo), nela vão ser criados os gameobjects envolvidos, a classe
deve conter 5 métodos, um método construtor, um método para carregar os
objetos envolvidos, um método para inicializar esses objetos, um método para
desenhar os objetos e outro método que representa o loop infinito do jogo. O
método construtor deve ficar responsável por chamar os outros métodos.
* @method 4
* @comment A classe deve conter 5 métodos, , um método construtor, um método
para carregar os objetos envolvidos, um método para inicializar esses
objetos, um método para desenhar os objetos e outro método que representa o
loop infinito do jogo. O método construtor deve ficar responsável por chamar
os outros métodos.
* @gameobject 3
* @comment A classe deve definir 3 gameobjects: Barra, Bola e Bloco.
* @window
* @comment Você deve atualizar a janela do jogo através do método update()
*/
public classe Jogo{
.............
```

Figura 11 - Marcadores de padronização da classe *Jogo*

A classe *Principal* define somente a instanciação de um objeto do tipo da classe *Jogo*, iniciando assim a execução do programa. Por este motivo a classe *Principal* também não foi inserida nesta análise de resultados.

Na análise do jogo são verificados os resultados das classes *Bola, Barra e Jogo*. Os resultados são quanto às análises gerais e detalhadas destas classes. As classes são avaliadas quanto a padronização do modelo e comparação de pares de variáveis (árvores de comportamento).

Entre os dez programas analisados, os estudantes 3, 5, 8, 9 e 10 estão com o comportamento geral do programa incorreto na primeira tentativa de desenvolver o exercício, conforme descrito na Tabela 4.

Tabela 4 - Relação dos programas com comportamento geral incorreto

ESTUDANTE	COMPORTAMENTO DO JOGO INCORRETO
3	O movimento da barra não é realizado através do controle do teclado, ao invés disso a barra segue a direção do objeto bola, sem que o usuário tenha qualquer tipo de controle através do teclado.
5	Não realiza o movimento da bola corretamente, ela fica presa no lado superior da janela.
10	Todos os objetos estão sem movimento.
8	O objeto bola não se movimenta para os lados corretamente. O movimento do objeto barra, apesar de ser controlado pelo teclado, também segue para o centro da janela do jogo sempre que a bola alcança uma determinada posição central na janela.
9	O movimento está incorreto, a barra é controlada através do uso do teclado somente quando o comando do teclado segue a mesma direção da bola, e está se movimentando no eixo "y", sendo que foi solicitado que a barra se movimentasse somente no eixo "x".

6.1 Resultados Obtidos por meio da Medição de Sensibilidade, Especificidade, Acurácia e Precisão

Da mesma forma que o estudo de caso 1, foi verificado dois tipos de resultados: o primeiro baseado por unidade de programa analisado e o segundo baseado na quantidade de comentários apresentados por programa.

As Tabelas 5, 6 e 7 apresentam, respectivamente, os resultados das medidas de eficiência das classes *"Bola"*, *"Barra"* e *"Jogo"* obtidas por meio do primeiro tipo de resultado (baseado por unidade de programa analisado).

As Tabelas 8, 9 e 10 apresentam, respectivamente, os resultados das medidas de eficiência das classes *"Bola"*, *"Barra"* e *"Jogo"* obtidas por meio do segundo tipo de resultado (baseado na quantidade de comentários apresentados por programa).

Tabela 5 - Medida de Eficiência da classe "Bola" de acordo com o total de programas analisadas para o segundo estudo de caso sugiro

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de programas analisados***		
Classe "Bola"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*1*	*1*
	FN	*2*	*1*
	PP	*7*	*8*
	FP	*0*	*0*
	Sensibilidade	*78%*	*89%*
	Especificidade	*100%*	*100%*
	Acurácia	*80%*	*90%*
	Preditividade Positiva	*100%*	*100%*
	Preditividade Negativa	*33%*	*50%*

Tabela 6 - Medida de Eficiência da classe "Barra" de acordo com o total de programas analisadas para o segundo estudo de caso sugiro

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de programas analisados***		
Classe "Barra"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*3*	*3*
	FN	*7*	*7*
	PP	*0*	*0*
	FP	*0*	*0*
	Sensibilidade	*0%*	*0%*
	Especificidade	*100%*	*100%*
	Acurácia	*30%*	*30%*
	Preditividade Positiva	-	-
	Preditividade Negativa	*30%*	*30%*

Tabela 7 - Medida de Eficiência da classe "Jogo" de acordo com o total de programas analisadas para o segundo estudo de caso sugiro

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de programas analisados***		
Classe "Jogo"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*5*	*9*
	FN	*4*	*0*
	PP	*1*	*1*
	FP	*0*	*0*
	Sensibilidade	*20%*	*100%*
	Especificidade	*100%*	*100%*
	Acurácia	*60%*	*100%*
	Preditividade Positiva	*100%*	*100%*
	Preditividade Negativa	*56%*	*100%*

Tabela 8 - Medida de Eficiência da classe "Bola"de acordo com o total de comentários analisados para o segundo estudo de caso

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de comentários analisados***		
Classe "Bola"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*3*	*12*
	FN	*2*	*6*
	PP	*7*	*8*
	FP	*0*	*0*
	Sensibilidade	*78%*	*57%*
	Especificidade	*100%*	*100%*
	Acurácia	*83%*	*77%*
	Preditividade Positiva	*100%*	*100%*
	Preditividade Negativa	*60%*	*67%*

Tabela 9 - Medida de Eficiência da classe "Barra" de acordo com o total de comentários analisados para o segundo estudo

de caso

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de comentários analisados***		
Classe "Barra"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*4*	*8*
	FN	*14*	*28*
	PP	*0*	*0*
	FP	*0*	*0*
	Sensibilidade	*0%*	*0%*
	Especificidade	*100%*	*100%*
	Acurácia	*22%*	*22%*
	Preditividade Positiva	-	-
	Preditividade Negativa	*22%*	*22%*

Tabela 10 - Medida de Eficiência da classe "Jogo" de acordo com o total de comentários analisados para o segundo estudo de caso

Projeto "BrickBreak"	***Medidas de eficácia do sistema analisador de acordo com o total de comentários analisados***		
Classe "Jogo"	***Quantidades e Medidas Obtidas***	***De acordo com a Padronização do Modelo***	***De acordo com a Comparação de Pares de Variáveis e Variáveis sem Par***
	NN	*6*	*37*
	FN	*6*	*33*
	PP	*1*	*1*
	FP	*0*	*0*
	Sensibilidade	*14%*	*3%*
	Especificidade	*100%*	*100%*
	Acurácia	*54%*	*54%*
	Preditividade Positiva	*100%*	*100%*
	Preditividade Negativa	*50%*	*53%*

7 Discussão dos Resultados

Durante o primeiro estudo de caso, percebeu-se que a análise mediante a padronização do modelo não apresentou resultados do tipo falso-negativos. Todos os resultados foram do tipo negativo-negativo, significando que eles apontam para um problema real de implementação do programa do estudante. Logo, o percentual de acerto da análise de acordo com a padronização foi de 100%.

A análise que mais apresentou resultados falso-negativos foi a comparação de pares de variáveis (comparação de árvores de comportamento) e variáveis sem par. A análise mostra um total de 38 (trinta e oito) comentários, sendo 9 do tipo falso-negativos (como mostra a Tabela 3). Verificando separadamente os resultados dos pares de variáveis e variáveis sem par, o percentual de erro, de acordo com a comparação dos pares de variáveis, foi de aproximadamente 33% e o percentual de acerto foi de aproximadamente 67%. E o percentual de erro, de acordo com a comparação das variáveis sem par, foi de aproximadamente 17% e o percentual de acerto foi de aproximadamente 83%.

Os erros de análise, neste caso, foram decorrentes da falta de padronização dos programas dos estudantes de acordo com o programa modelo do professor. É possível concluir que a comparação de variáveis se mostra mais suscetível a erros de análise. Isto acontece porque os erros de análise decorrentes da comparação de variáveis são consequência principalmente das variações de programação e da falta de padronização do programa do aluno de acordo com o programa modelo do professor. Quanto mais padronizado o código do aluno estiver, menos erros de análise decorrentes de variações de programação serão apresentados.

A padronização do programa são sugestões para o aluno sobre a construção de código que, em uma primeira etapa de construção, apesar de poder ser considerada como diminuidor da criatividade do aluno, pode ser também um fator muito relevante para um primeiro entendimento e descobrimento da solução do problema, permitindo assim, através dessa primeira experiência de solução, construir outros caminhos de descoberta na solução do mesmo e de novos problemas. De certa forma, a padronização pode ser vista como requisitos do jogo.

A precisão de acerto da comparação de variáveis depende do estudante seguir corretamente as orientações de padronização repassadas pelo professor na atividade. A precisão de acerto, neste caso, também depende da forma como o professor padroniza o programa modelo, sendo importante que o professor identifique quais comportamentos precisam de padronização ou não. O professor deve verificar quais são os pontos do programa que são mais sujeitos a variações de programação e se estes pontos são comportamentos relevantes para a execução correta do programa do estudante. Em caso de serem comportamentos relevantes, o professor deve estabelecer orientações de padronização bem definidas para o aluno e comentar os comportamentos no código do programa modelo. Se não forem comportamentos relevantes, o professor pode optar por não definir padronização, bem como não comentar estes pontos do programa evitando impressão de comentários falso-negativos desnecessários.

Na análise de variáveis sem pares, 50% dos alunos apresentaram resultados falso-negativos. No entanto, todos os alunos apresentaram o mesmo comentário falso-negativo, retratado no comentário 5. Em consequência do resultado recorrente falso-negativo deste comentário, pode-se concluir que o professor poderia optar por não comentar este comportamento, já que existem variações de programação possíveis para o mesmo. Esta análise demonstra um nível mais elevado de falta de padronização (orientações repassadas pelo professor) do programa do aluno.

Já para o segundo estudo de caso foi identificado que, de acordo com a padronização, somente a classe *Bola* foi aprovada, ou seja, a hipótese h1 (PNN>PFN) foi comprovada. Quase todas as medições estatísticas, no caso da classe *Bola* foram superiores a 50%. Somente a preditividade negativa para o cálculo baseado na quantidade de classes de programas é inferior a 50%. No entanto, a mesma medida baseada na quantidade de comentários é superior a 50%. A classe *Barra* e a classe *Jogo*, no entanto, foram reprovadas, ou seja, no caso delas, a hipótese h1 não foi aceita. Pode-se verificar que o alto índice de falta de padronização nas classes *Barra* e *Jogo* influenciam os resultados da comparação na próxima fase de análise (comparação de variáveis), conforme se pode observar nas Tabelas 11 e 12. A classe *Barra* indica falha na padronização da classe quanto a hipótese 1 (problemas identificados pelos marcadores *@Keyboard* e *@method*). Por sua vez, a falha na padronização compromete a comparação de variáveis, gerando altos índices de falso-negativos nesta fase. Por motivo da falta de padronização, a acurácia e a preditividade negativa, na classe *Barra*, apresentam valores inferiores a 50%.

A classe *Jogo* também apresenta falha na padronização da classe quanto à hipótese 1, com um resultado de PNN=PFN. Quanto a comparação de variáveis em relação à classe alcança-se uma situação de aprovação, mas com uma taxa de PFN muito próxima a PNN.

Tabela 11 - Medida de Eficiência do Sistema Analisador de acordo com o total de comentários analisados para o segundo estudo de caso

Classe	**Avaliação dos resultados quanto à padronização e comparação de variáveis nas classes do jogo**					
	PADRONIZAÇÃO QUANTO Hipótese 1			**COMPARAÇÃO DE VARIÁVEIS QUANTO Hipótese 1**		
	Status da classe	***PNN***	***PFN***	***Status da classe***	***PNN***	***PFN***
"Bola"	Aprovado	60%	40%	Aprovado	67%	33%
"Barra"	Reprovado	22%	78%	Reprovado	22%	78%
"Jogo"	Reprovado	50%	50%	Aprovado	53%	47%

Tabela 12 - Avaliação dos resultados quanto à padronização e comparação de variáveis nas classes do jogo

Classes	**Avaliação dos resultados quanto à padronização**					
	QUANTO À QUANTIDADE DE CLASSES DE PROGRAMAS			**QUANTO A QUANTIDADE DE COMENTÁRIOS**		
	Sensibilidade	***Acurácia***	***Preditividade Negativa***	***Sensibilidade***	***Acurácia***	***Preditividade Negativa***
"Bola"	78%	80%	33%	78%	83%	60%
"Barra"	- *<<não apresentou valores PP e FP>>*	30%	30%	- *<<não apresentou valores PP e FP>>*	22%	22%
"Jogo"	20%	60%	56%	3%	54%	53%
Classes	**Avaliação dos resultados quanto a comparação de variáveis**					
	QUANTO A QUANTIDADE DE CLASSES DE PROGRAMAS			**QUANTO A QUANTIDADE DE COMENTÁRIOS**		
	Sensibilidade	***Acurácia***	***Preditividade Negativa***	***Sensibilidade***	***Acurácia***	***Preditividade Negativa***
"Bola"	89%	90%	50%	57%	77%	67%
"Barra"	- *<<não apresentou valores PP e FP>>*	30%	30%	- *<<não apresentou valores PP e FP>>*	22%	22%
"Jogo"	100%	100%	100%	3%	54%	53%

A classe *Jogo* inclui os objetos necessários para a criação do cenário do jogo, controle por meio do teclado e instanciação dos objetos do tipo *Bola* e *Barra*. Todos os comportamentos destes objetos devem ser executados, bem como as regras do jogo devem ser implementadas no *gameloop* (*loop* infinito) desta classe. Por ser uma classe mais detalhada, que envolve as outras classes do programa, a descrição da padronização e das regras do jogo demanda mais dificuldade de descrição do que as outras classes e deve ser feita de maneira minuciosa pelo professor. O alto índice de falso-negativos na classe *Jogo* faz a medição da sensibilidade ficar abaixo de 50%. Já os valores acurácia e preditivo negativo se mantiveram iguais ou um pouco acima de 50%. Na classe *Jogo*, mais uma vez, verifica-se que problemas de padronização comprometem os resultados na próxima fase de comparação (comparação de variáveis).

Somente a classe *Bola* indica aprovação nas duas fases da análise (padronização da classe e comparação de variáveis). Quanto a hipótese 1, a classe *Bola* também apresenta os melhores resultados quanto as medidas estatísticas.

Como a comparação de variáveis depende diretamente de uma padronização adequada do programa do aluno de acordo com o programa modelo do professor, sugere-se que os problemas de padronização sejam apontados pelo analisador ao aluno e que a segunda fase de comparação de variáveis só seja realizada mediante um resultado de padronização que não apresente reprovações.

8 Conclusão e Trabalhos Futuros

É possível concluir, de acordo com a análise de resultados realizada nos dois estudos de casos, que quanto mais padronizado o programa do estudante em relação ao programa modelo, menos imprecisões serão encontradas na comparação (o que significa uma menor ocorrência de resultados do tipo Falso-Negativos). Ainda no caso do comportamento do programa estar correto, mas não se encontrar de acordo com a padronização, a falta de padronização vai gerar comentários falso-negativos. Uma alta incidência de falso-negativos na padronização da classe compromete a eficiência da próxima fase de comparação do analisador, a comparação de pares de variáveis, gerando muitos resultados falso-negativos, também, na próxima fase.

Problemas de padronização podem ser causados pelo aluno, no caso dele não ter seguido as orientações corretamente, mas também podem ser causados pelo mau detalhamento ou má escolha (quando existem muitas variações de programação) do professor quanto aos comportamentos da classe que devem ser padronizados. Uma vantagem do sistema analisador é que o próprio professor pode identificar pontos críticos que causam problemas na padronização dos programas e pode alterar, no programa modelo, os comportamentos padronizados, de forma a corrigir e/ou minimizar esses problemas.

Resultados do tipo falso-negativos, em geral, são ocasionados pelo fato de existirem muitas variações de programação para resolver o mesmo problema. Neste caso, o professor deve padronizar, no programa modelo, o modo de solucionar o problema, e o aluno deve seguir as orientações de acordo com a padronização do modelo. O professor ainda pode escolher não associar nenhum comentário ao comportamento envolvido (no caso do professor verificar que o comportamento não modifica o comportamento geral do programa e/ou não é relevante para avaliação naquela atividade específica), liberando o aluno para uma implementação mais subjetiva (desta forma as diferenças são computadas, mas não existem comentários associados a serem enviados para o aluno). Mais uma vez, o sistema analisador apresenta a vantagem de o próprio professor, por meio de sua verificação durante a aplicação da atividade, poder modificar as configurações do programa modelo e assim corrigir e/ou minimizar os problemas apresentados.

Conclui-se também que dois tipos de critérios podem ser utilizados para a classificação de tipos Falso-Negativo e Negativo-Negativo: o primeiro critério diz que uma diferença detectada é do tipo Falso-Negativo quando ela não modifica o comportamento correto do programa e um Negativo-Negativo quando a diferença encontrada modifica o comportamento correto do programa (foi o critério utilizado nos estudos de casos). O segundo critério possível diz que uma diferença detectada é do tipo Falso-Negativo somente quando ela não identifica uma diferença real de padronização entre os dois programas ou quando ela não identifica uma diferença real entre pares de variáveis e um Negativo-Negativo é identificado toda vez que uma diferença real de padronização ou de pares de variáveis for encontrada, independentemente desta diferença modificar ou não o comportamento correto do programa. O segundo critério, tal como pode ser visto nas Tabelas 8 e 9, se utilizado, é capaz de causar uma melhora significativa nos resultados de medidas de eficiência do sistema *JPlay* Tutor (acurária, preditividade negativa etc). Neste trabalho não se definiu qual dos critérios é o mais adequado

para medir a eficiência do analisador, de forma que o estudo e escolha do critério mais adequado tornam-se relevante em trabalhos futuros.

Ainda para trabalhos futuros torna-se necessária a construção de uma interface de tutoria capaz de interpretar os resultados obtidos e, então, a partir disso, mostrar as respostas para o estudante. A interface de tutoria ficaria encarregada, dentre outras funções, por verificar os resultados na fase de padronização do modelo, de maneira que o analisador somente passaria para a próxima fase de análise (fase de comparação de pares de variáveis) se o programa do aluno estivesse corretamente padronizado. Outra alternativa seria a de que os índices de aceitação h1 para a análise da próxima fase pudessem ser configurados previamente pelo professor, de forma que ele possa controlar o quão padronizado deve estar o programa do estudante para que seja permitida a análise da próxima fase (comparação de variáveis).

Para trabalhos futuros também se propõe realizar estudos de casos com a participação de mais de um professor, de forma a avaliar o comportamento do professor no uso do analisador e se diferentes professores poderiam interferir nos resultados da análise desenvolvida.

Acredita-se que esta proposta traz contribuições significativas na área de programação e ensino de programação, tendo como principal contribuição, uma heurística baseada na comparação de comportamentos entre programas (programa do aluno e programa modelo). A proposta contribui, também, com a implementação de uma ferramenta capaz de interpretar o código semanticamente construído por programadores, retornando resultados, apontando problemas e sugerindo soluções semânticas.

Referências

ADAM, Anne; LAURENT, Jean-Pierre. LAURA*, a system to debug student programs. Artificial Intelligence*, v. 15, n. 1-2, p. 75-122, 1980.

ALEXIS, V. de A.; FERREIRA, Deller J. Aplicando padrões de seleção no ensino de programação de computadores para estudantes do primeiro ano do ensino médio integrado. X Encontro Anual de Computação–EnAComp, 2013.

ALLEN, Eric; CARTWRIGHT, Robert; STOLER, Brian. DrJava: *A lightweight pedagogic environment for Java*. In: ACM SIGCSE Bulletin. ACM, 2002. p. 137-141.

ALLOWATT, Anthony; EDWARDS, Stephen H. *IDE Support for test-driven development and automated grading in both Java and C++*. In: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange. ACM, 2005. p. 100-104.

BARBOSA, Leônidas S.; FERNANDES, Teresa CB; CAMPOS, André MC. Takkou: uma ferramenta proposta ao ensino de algoritmos. In: XVIII Workshop sobre Educaçao em Computaçao (WEI 2011). 2011.

SANTOS, E.C.O., BATISTA, G.B., SOUSA, V.H.V., CLUA, E.W.G. *" A Semantic Analyzer for Simple Games Source Codes to Programming Learning". In SEKE 2014: The Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering*, p. 522-527, Vancouver, Canada, *July 1-3*, 2014a.

SANTOS, E.C.O., BATISTA, G.B., SOUSA, V.H.V., CLUA, E.W.G. *"Structural Analysis for Simple Games Source Codes Applied to Programming Learning".* In SBGames 2014: XIII Simpósio Brasileiro de Jogos e Entretenimento Digital, p. 71-79, Porto Alegre, 2014b.

BOTELHO, C. A. "Sistemas Tutores no domínio da programação". Revista de Informática Aplicada/*Journal of Applied Computing*, v.4, n. 1, 2010.

DELGADO, K. V. "Diagnóstico baseado em modelos num sistema inteligente para programação com padrões pedagógicos". *Master's dissertation, Institute of Mathematics and Statistics*. 2005.

DIFFEN, disponível em http://www.diffen.com/difference/Accuracy_vs_Precision. Acessado em novembro de 2015.

FEIJÓ, B., CLUA, E., Da SILVA, F.S.C. Introdução à Ciência da Computação com Jogos: Aprendendo a Programar com Entretenimento. Campos Elsevier.1º ed. 2010.

HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J.H. "*The elements of statistical learning data mining, inference, and prediction*. New York: Springer, 2009.

JOHNSON, W. L., SOLOWAY E. "*Proust: Knowledge-based program understanding". In ICSE 84: Proceedings of the 7th international conference on Software engineering*, pp. 369-380, Piscataway, NJ, USA, 1984. IEEE Press.

JPLAY, *available in* http://www.ic.uff.br/jplay/. *Accessed in April* 2012.

KOLLING, M., QUIG, B., PATTERSON, A., and ROSENBERG, J. "*The BlueJ system and its pedagogy". Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology*, v.13, n.4, p.249-268,2003.

PINHEIRO, W.R., BARROS, L.N., Kon, F. "AAAP: Ambiente de Apoio ao Aprendizado de Programação". In Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação, São Paulo, 2007.

RUN.CODES, disponível em https://run.codes/. Acessado em janeiro de 2016.

SANTOS, N.S.R.S., RAPKIEWICZ, C.E. "Ensinando princípios básicos de programação utilizando jogos educativos em um programa de inclusão digital". In: SBGAMES - VI Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital, 2007, São Leopoldo - RS.

TRAETTEBERG, H., AALBERG, T. "*Jexercise: a specification-based and test-driven exercise support plugin for eclipse". In eclipse "06:Proeedings of 2006 OOPSLA workshop on eclipse technology eXchange, pages* 70-74, New York, NY, USA. ACM Press. 2006.

WAINER, J. "Métodos de pesquisa quantitativa e qualitativa para a ciência computação". In: Atualização em Informática, Tomasz Kowaltowski and Karin Breitman. (Org.), Sociedade Brasileira de Computação e Editora PUC-Rio, 2007.

WIKIHOW, disponível em http://pt.wikihow.com/Calcular-Sensibilidade,-Especificidade,-Valor-Preditivo-Positivo-e-Valor-Preditivo-Negativo. Acessado em novembro de 2015.

Submetido para avaliação em 21 de novembro de 2016
Aprovado para publicação em 22 de março de 2017

Elanne Cristina Oliveira Dos Santos
Instituto Federal de Educação, Ciência e Tecnologia do Piauí – IFPI, elannecristina.santos@ifpi.edu.br

Gleison Brito Batista
Universidade Federal de Minas Gerais – UFMG, gleisonbrito@dcc.ufmg.br

Esteban W. Gonzales Clua
Universidade Federal de Fluminense – UFF, Niterói, Brasil, esteban@ic.uff.br