A Semantic Analyzer for Simple Games Source Codes to Programming Learning

Elanne Cristina Oliveira dos Santos\textsuperscript{1,2}
Victor Hugo Vieira de Sousa\textsuperscript{1}
\textsuperscript{1}Instituto Federal de Educa\c{c}\~{a}o, Ci\c{c}\~{c}encia e Tecnologia do Piau\~{i}, IFPI
Teresina, Brasil
elanecristina.santos@ifpi.edu.br
vhv.sousa@gmail.com

Gleison Brito Batista\textsuperscript{2}
Esteban Walter Gonzales Clua\textsuperscript{2}
\textsuperscript{2}Universidade Federal de Fluminense, UFF
Niter\c{o}, Brasil
gbatista@ic.uff.br
esteban@ic.uff.br

Abstract— The teaching of programming and algorithms consists in a big challenge, not only in universities but also in schools and training centers. Many proposals for stimulating this process were made in the last years. Previously to this work we proposed a semantic analyzer based on behaviors comparison between two programs, the model program and a student program. The programs compared are simple 2D games developed by JPlay framework. The JPlay was proposed and developed for teaching programming and consist on a framework that facilitates the teaching of programming. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar pairs of classes. While in a previous work we proposed how to combine similar classes of two programs, in this paper we show how to analyze similar classes, based on a variable behavior strategy. These behaviors are identified based at an occurrence scope. Therefore, when behaviors differences are identified, our proposed system makes semantical suggestions about these behaviors differences found in the student program. The teacher adds and edits the suggestions using comments in the source code of the model. In this paper, we will also show some results of this comparison.

Keywords- programming; JPlay; knowledge modelling; classification; code comparison; teaching; games

I. INTRODUCTION

The teaching of programming and algorithms consists in a big challenge, not only in universities but also in schools and training centers. Studies point to the difficulty of teaching and learning the disciplines related to algorithms and programming, resulting in high dropout rates in computer courses [13][5]. The main reason for this negligence is the difficulty in learning abstract concepts of programming [16]. Many proposals for stimulating this process were made in the last years [3][12][17][4].

In this sense, the JPlay framework was proposed and developed for teaching programming [9][11]. JPlay is a framework for facilitating the teaching of programming, providing an algorithmic learning process related with the logic of simple 2D game development. JPlay does not interfere with the structure of basic programming necessary for a correct learning of algorithmic logic and does not introduce specific features of design patterns or stereotypes of games in the source code. The tool allows the students an easy way to draw and move images on a computer screen and provides methods and objects that help to create 2D games using the Java language.

Previously to this work we proposed a semantic analyzer based on behaviors comparison between two programs, the model program and a student program. The programs compared are simple 2D games developed by JPlay framework. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar pairs of classes. Previously, we show how the algorithm combined the similar classes of the two programs [15]. In this paper we will show how the similar classes will be analyzed. This analysis will show more specifics results using a comparison algorithm between the variables of same type of each classes pair, analyzing its game context instead syntax details and based on variables behaviors. Behaviors are identified through the analysis of its occurrence scope at the source code, such as an assignment, a loop or a conditional usage. Therefore, when behaviors differences are identified, the system makes suggestions about these behaviors differences found in the student program and clues that may indicate a possible semantic error. The teacher adds the suggestions in the form of comments in the source code of the model program, which are automatically captured by our solution. In this paper, we will also show some results of this comparison.

II. RELEATED WORK

During the process of learning programming different techniques can be used for students in addition of learning to program with the purpose of acquiring good programming practices. The techniques are classified as follows: tests based programming, programming pattern, automatic evaluator, programs diagnostic systems [13].

Based on previous works [7][13], we classify the techniques related to programming learning context as follows: programming based unit testing, proposals of programming
environments, automatic evaluator, analysis of programming patterns and automatic depuration systems (intelligent tutoring systems and programs diagnostic systems).

In programming based unit the teacher provides a set of specific tests to solve a particular problem and the student must build a program that allows the achievement of expected results in the execution of all tests [13][17]. The proposals of programming environments consist on the fact that some development tools were created in order to assist students in introductory programming, such as BlueJ and DrJava [12][3].

The automatic evaluator is used to help the teachers with tasks of activities corrections. The teacher can define acceptance tests to be automatically executed after the students deliver their programming activities and results of the tests can be used to compose the final score the student [13]. We can quote the Web-Cat [4] as an automatic evaluation tool.

The programming of analysis patterns is based on research of programming learning suggestions that experienced developers solved when looking for previous solutions that are related to the new problem and that can be adapted to the ideal situation [7]. Thus, the concept of patterns is based on the fact that experienced programmers are able to solve new problems through the analysis of a previously solved problem. They can identify what structure to use, what types of data is involved, as well as other ways to solve the same problem, through previous experiences that identify solutions [2]. Previous experiments contain the basis for Programming patterns, which are solutions that often appear in solving computational problems [2]. Thus, patterns translating programming strategies created by experts can lead to good programming practices. We can quote the systems Proust [10] and PROPAT [7] as systems that use the strategy of analysis of programming patterns.

An automatic depuration system is a system that uses techniques in order to find and classify components from a program. Based on the type of technique used, this may be classified as a programming intelligent tutoring systems and program diagnostic system.

Laura is one of the first attempts to build a tutoring system for teaching programming and is written in Fortran [6]. Its strategy is a comparison between two programs, the model and the candidate. The comparison is possible through the representation of the model and candidate programs by graphs, and its heuristic strategy to identify step by step the elements of the graphs [1].

We have previously developed a knowledge modeling system for semantic analysis of Games [15] and is based at learning objectives. It aims to find and to classify possible errors that happen in the program. Therefore it can be used in order to guide the student about these errors. It has a function of interpreting semantically and architecturally a Java program developed that uses the JPlay and return results of this examination to the programmer. The process consists on a comparison between the student program and model program. For this, the programmer must select, in his integrated development environment (IDE) tool, the model program that he wants to use as reference, previously available in a repository. Thus, the analyzer is able to interpret semantically the program that is being built by the student, may point out problems and suggest possible solutions.

The comparison is based on behaviors of the programs. Different behaviors between the model program and student program produce suggestions about possible errors in the student’s source code. Thus, in order to compare the behaviors of two programs we developed a comparison algorithm between the classes of the model program and student program, resulting in similar classes pairs. Previously, we show how the algorithm combined the similar classes of the two programs [15].

After the selection of similar classes, variables of each pair are compared and similar variables pairs are selected. Then, at this stage, the analysis shows more specifics results using a comparison algorithm between the variables of the same type of the pairs of classes. The comparison is based on variables behaviors. Behaviors are identified through the occurrence, at the source code, of assignments, loops or conditionals statements. Therefore, when behavioral differences are identified at the student program, the algorithm makes suggestions about. The teacher adds these suggestions in the form of comments in the source code of the model program.

A difference is that our proposal is based on a design pattern oriented to simple 2D game, following the original purpose of JPlay [9][11]. One more important difference concerns the comparison between similar variables. In the case of Laura, for example, two graphs of the model program and candidate program are built and compared. In our work we can build a data structure (behavior tree) starting from the behaviors of the variable, and compare each of these structures, thus obtaining a higher level of granularity in this heuristic strategy in order to identify behaviors differences between programs.

III. JPlay

Our proposal is based in the JPlay framework. JPlay was previously developed with the purpose of teaching computer science and algorithms based in game development. In order to identify sequential patterns in JPlay architecture, we divide the JPlay diagram into three parts: the interaction between game and player, characters and output game.

The classes responsible for interaction between game and player are: Keyboard (define input data for the keyboard) and Mouse (define input data for mouse). The classes responsible for creating the characters of the game are: Animation (defines an animation. It must have a picture and their frames. A frame is a piece of the picture responsible for the movement of animation), Sprite (the Sprite class extends the Animation class. The Sprite class contains methods that can make the image move across the screen) and Body (the Body class extends the Animation class. Like Sprite, the Body class also contains methods that can move the image, and beyond these methods it adds methods to accelerate and decelerate the image across the screen). The classes responsible for outputs in the game are: Window (defines a window where all the game elements will be drawn), Time (defines a time counter), Sound (defines the sound that will be played in the game) and Collision (it is a static class, used to check if there was a...
collision between two objects. The occurrence of a collision can be verified using this method in all classes, or by the Collision static class).

Our proposal is based on design patterns of sequence used at the JPlay framework. JPlay follows a typical game framework pattern: objects, also called as game objects, are initially defined. A loop is initiated (also called as a game loop) and each iteration corresponds to a frame being produced. In this loop all game objects are updated with their corresponding logic (coming from an AI algorithm, physic algorithm or even from the user interface sequence). Finally, all the elements of the game are drawn in the screen.

IV. SEMANTIC ANALYZER ARCHITECTURE

We propose in this work a semantic analyzer for the code being generated by the students. The system is composed by many stages and modules, which are illustrated in Fig. 1.

Basically, the system starts classifying pairs of classes that are taken from both the model class, which corresponds to the model program (teacher’s program), and student code (1). The analyzer then checks if the pairs of classes are standardized according to the properties defined by the teacher for each exercise, which corresponds to the model program (2).

Thus, in the standardization phase (2), the student’s code must be standardized according to the model program so that they can be compared later. Thus, a markers structure must be filled in each of the classes of the program model. Each marker indicates the characteristics that the class must have to implement their behavior. The teacher must inform the values of the markers in each of the classes of the program model. Then, the analyzer identifies, in the student code, the values of the markers of each class. Some of the markers that are used are: inheritance, constructor, movex, movey, keyboard, mouse, method, game object, main and game loop. Although we define basic markers, more specific markers can be defined by the teacher. If the standardization is correct for each pair of classes, the analyzer starts the comparison process (4), classifying variables pairs between pair of classes, otherwise the analyzer requests the student adaptations in the code and the process returns to the beginning.

In the case of the JPlay Sequential Pattern (3), if the code is correct (a sequence code pattern of the JPlay is a code sequence in the program based on JPlay that must always happen when the program is correct), the analyzer starts the process (4), otherwise the analyzer requests the student adaptations in the code and the process returns to the beginning. Thus, after process (4), the comparison of behavior trees of each variables pairs is performed (5) and the differences between the programs is returned to the student.

V. MODELING AND ANALYSIS OF PAIRS OF SIMILAR CLASSES BETWEEN PROGRAMS

Our proposal in [15] affirm that since Badros approach allows the preservation of the source code and our method needs a subsequent semantic analysis, we initially convert all classes from a developed program into a XML representation, based on the proposed JavaML method. Due the increase of tag’s representation from JavaML 2.0, we ignored this update. In order to semantically analyze a code under development, the programmer must select in a repository another program which will act as a model program for the comparison. The analysis consists in compare pairs of classes. Every class of the programmer code and from the program base will initially be transformed into XML by the parser. Each XML file will be read and interpreted by Java language using Document Object Model (DOM) [8].

After defining the pairs of classes, the comparisons between pairs are performed. The goal on this stage consists on identifying the pairs of classes that have higher similarity. The analysis of similarity between pairs of classes is based on rules previously established [15]. The matrix is first initialized with its similarity values, represented by the "weight" attribute, equal to 0 (zero) in all its elements. Other attributes are initialized with null.

There are specific rules to find the first and second pair. In order to classify the first pair of similar classes we establish as a single rule that the class contains the “main” method. When it finds two classes containing this method, the weight value is 2 and the “main” attribute value is true. After, in order to classify the second pair of similar classes we establish only one rule: if the class contains the game loop. When it finds two classes containing the game loop, the weight value is 2 and the "gameloop" attribute value is true. The others attributes are calculated according to the rules defined in [15] in order to be performed more analysis of the student program.

After the first and second pair of similar classes be classified, the analyzer will take the rest of the classes of the program to be processed. This classification is performed in levels. First, the analyzer will search pairs of classes that extend from the same super class. In Fig. 2, for example, the “MyBall” class belongs to student program and the “Ball” class belongs to model program. During the classification the analyzer verifies that the “MyBall” and “Ball” classes extend from the same Sprite class. The analyzer verifies the list of variables of each one of the classes of the program that extend the same super class. In Fig. 3, for example, the “MyBall” and the “MyBar” class belong to the student program and extend Sprite class, the “Ball” and “Bar” class belong to model

![Figure 1. Stages of the Semantic Analyzer Process](image-url)
program and also extend Sprite class. Then, all the combinations between these classes of the two programs will be analyzed with the goal of finding the pairs of classes more similar. In this step the combinations are performed by comparing the lists of variables of each class from the programs, as shown in Fig. 4. Each variable list contains the variable type and the number of variables of each type. The algorithm compares each pair of lists and calculates the difference between the values of variables of same type, after the similarity weight assigned the value of the sum of the results. Then, the pair having the lowest weight is rated as the most similar pair.

Generally the comparison between the lists of variables each class does not get full precision. Thus, the results may be close to reality, but not entirely correct. In order to obtain greater accuracy in the result, the algorithm performs a second comparison based on lists of behaviors. At the next comparison, the algorithm generates lists of behaviors containing the type of behavior and the value of that type of behavior. The algorithm compares each pair of lists again and then the pair containing the smallest weight is classified as most similar pair. At this level, all pairs of similar classes are defined, and the pairs classified should be compared.

At this point, with the aim of comparing the behavior of the pairs of classes, variables of the same type in each of the classes of the pair should be compared according to their behavior and then pairs of variables should be classified. The algorithm compares the variables of the same type using a list of behaviors and then the pair containing the smallest weight is rated as the most similar pair of variables.

After that, similar pairs of variables are defined and then analyzed. In order to compare these variables, we propose a tree structure, called behavior tree. This structure contains all the behaviors of a variable, which in our case we defined as “assignment”, “conditional” and “loop”. We build a behaviors tree to each variable, and the analyzer compares the behavior tree of a variable of a class that belongs to the student program with the behavior tree of its similar variable from model program. The behavioral trees of both the variables are compared, thus when behaviors differences occurs it is possible that a probable error of the student program occurs. In this point, when behaviors differences are identified in the trees, the analyzer makes suggestions about these behaviors differences found in the student program and clues that may indicate a possible semantic error. The suggestions are defined as comments in the source code of the program model. Each comment must be entered by the teacher before each behavior of variable, thus are automatically captured by our solution as suggestions to student.

An example of the comparison between variables behavior trees is shown in Fig. 5 and Fig. 6. The comparison between the behaviors trees of variables “b” and “d” show a difference in the while behavior. The analyzer must find the difference of the conditions between the programs. Finally, it must send suggestions to the student describing all the differences detected between the behaviors of the classes.

VI. RESULTS

In this paper we propose a novel semantic analysis approach that checks a JAVA code and guide a student for a specific game development, giving clues of possible semantic failures, within a game oriented framework.
We developed a validation scenario in a classroom with 6 students of the integrated high school of the Informatics course. The students were proposed to develop a “BrickBreak” game, using JPlay framework. In this paper, we analyzed the “Ball” and “Bar” classes of the 6 codes developed, based on the model program developed by the teacher.

The results analysis presented here are related with two levels of analysis (according to Fig. 1): Checking for standardization between programs (2) and Comparison of the behavior trees (5). We evaluated the results according to behaviors expected for each class. The Ball class has two basic specifications: it must move in the x and y axis and collide with objects. The Bar class also has two specifications: it should move through the keyboard control, for both right and the left sides and collide with the Ball.

The results were marked as positive when the analyzer does not detect differences between the student program and the standardization model and negative when the analyzer detects differences between them. For the behavioral trees, we evaluated the results as false-positive when the analyzer does not detect differences, but the behavior of the object is not correct and false-negative when the analyzer detects differences but the behavior of the object is correct. It is considered, negative when the analyzer detects differences that really it exists and positive when the analyzer does not detect differences and the behavior of object is correct. Table I summarizes the results for the evaluation strategy for the Ball class example. Students 1 and 4 used one method more than requested in the standardization of model. The analyzer checks the difference between the students program and standardization model and prints the following suggestion for the student:

“You probably defined more methods than the necessary.”

About the comparison of behavior trees, there are differences in the program of the students 1, 3, 4, 5 and 6, according to Table I. The differences are defined as false-negative because these differences don’t modify the program behavior. Most of the differences in this class refer to assignment behaviors and are false-negative results.

Student 1 observations are related with the assignment behaviors of variables left and right from the model program and the suggestions that will be printed for him are:

- In the class constructor, initialize the attribute that controls the initial movement of the ball to the left or right;
- In the class constructor, initialize the attribute that controls the initial movement of the ball for up and down directions.
- Modify the movement of the ball if the ball position at the X axis is less than minimum limit of the game window and it is going to the left.
- Modify the movement of the ball if the ball position in the X axis is greater than maximum limit of the game window and it is going to the right. Consider this case the width of the ball. Example: maximum limit - width of the ball.
- Change the direction of the ball if the height of the ball in the y-axis is less than minimum limit of the game window and it is going to the left.
- Change the direction of the ball up if the height of the ball in the y-axis is greater than maximum limit of the game window and is going down.

The students 3, 5 and 6 show a similar situation to the student 1. For the student 4, the differences are related to the conditional behaviors of the variable “this.width” of the model program. The suggestion will be:

- Define the movement of the ball to the right side. Check which is the maximum right margin of the game window (maximum value of the x axis).

The results of the evaluation of the Bar class at the example is summarized by Table II:

<table>
<thead>
<tr>
<th>Student</th>
<th>Status program</th>
<th>Analysis according to the standardization between programs</th>
<th>Evaluation for Bar class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Incorrect (with the standard model)</td>
<td>Negative</td>
<td>Negative, False-negative</td>
</tr>
<tr>
<td>2</td>
<td>Correct</td>
<td>Positive</td>
<td>Positive</td>
</tr>
<tr>
<td>3</td>
<td>Correct</td>
<td>Positive</td>
<td>False-negative</td>
</tr>
<tr>
<td>4</td>
<td>Incorrect (with the standard model)</td>
<td>Negative</td>
<td>False-negative</td>
</tr>
<tr>
<td>5</td>
<td>Correct</td>
<td>Positive</td>
<td>False-negative</td>
</tr>
<tr>
<td>6</td>
<td>Correct</td>
<td>Positive</td>
<td>False-negative</td>
</tr>
</tbody>
</table>
The analyzer did not find occurrence of the key words “Keyboard” and “keydown” in the programs of all the students, according Tab. II. The standardization model has defined JPlay objects and methods with these names with the purpose of being used to implement the movement of the bar through the keyboard control. Students 1, 2, 4, 5 and 6 implemented the keyboard control in another class program. Some of them even used another JPlay method (called movex()) to accomplish the same behavior, that led to the indication of the errors, although the behavior of the programs were correct. Then the analyzer prints the following comment for the student:

- Please, check if you defined the movement of the bar through the keyboard control.

In the analysis of the behavior trees of students 1, 2, 4, 5 and 6 there are differences in all the behaviors of the variables of the Bar class comparing with the model program, because the analyzer could not mount pairs of similar variables, thus all the comments associated will be printed as simple suggestions for the student.

In the special case of the student 3 (three), the program behavior is actually incorrect, because the bar is not moving properly. Thus, it is not possible compare variables, form pair of variables and compare the behaviors trees. Then the following comments are made as suggestions:

- “Define the movement of the bar to the right through the use of the keyboard using the KeyDown() method. Check which is, in the game window, the maximum value of the right margin.”
- “Increase the movement of the bar on the x axis, making the bar moves to the right.”
- “Define the movement of the bar to the left through the use of the keyboard using the KeyDown() method. Check which is, in the game window, the minimum value of the left margin.”
- “Decrement the movement of the bar on the x axis, making the bar move to the left.”

VII. CONCLUSION

This paper presents a novel heuristic strategy based in a analyzer that, interpreting semantically a JPlay code, guide a student for a specific game development process. Although our implementations and tests are related to JPlay framework, our proposal can easily be adapted to other program patterns.

The goals of the analyzer are to interpret semantically a Java program that uses JPlay and return results of this analysis to the student. Our proposal brings significant contributions to researchers working in the field of programming education and software engineering, specifically in relation to the knowledge modeling, having as main contributions the architecture for classification of similar classes and the definition of the data structure (behavior tree) starting from the behaviors of variables. Our paper also contributes in the sense that introduces a tool able to semantically interpret code built by students, returning results, pointing out problems and suggesting solutions.

The results of the comparison of behavior trees show that the analyzer is not able to be totally accurate, but is able to make a suggestion closer to the truth to the student. The results of analysis according with the standardization between programs are most accurate, how much more standardized the student program, will be found less inaccuracy on comparison of the behavior trees.

As future work we intend to improve the efficiency of the algorithm and obtain more results comparisons. Also, as future work, we intend to develop a tutoring interface in order to manage the results received by the analyzer and the communication with the student.

REFERENCES