Reusing and Deriving Games for Teaching Software Reuse

Diego Castro, Cláudia Werner

COPPE/Computer Systems Engineering Program
Federal University of Rio de Janeiro
{diegocbcastro, werner}@cos.ufrj.br

Rio de Janeiro, Brazil

***Abstract*—Software Reuse (SR) is a crucial discipline that seeks to create new products using pre-existing artifacts. Companies are increasingly looking for members with these skills. However, they do not always find them. One of the possible causes is the lack of dedication of students in the classroom. To increase students' motivation and engagement in the classroom, many scholars are already using games as a teaching method, due to the various advantages that this strategy can bring to the current teaching method. SR can be related to several areas, such as Software Engineering, processes, or programming, more commonly associated with the latter. Based on this, the paper sought to identify games that aimed to teach programming but could also be used to teach the fundamentals of software reuse. Two games were created/derived from those found, and it was concluded that games could be derived or reused for more than one teaching context.**

Keywords–game; game-based-learning; software reuse; programming; systematic mapping.

I. INTRODUCTION

This work is an extension of the paper presented by Castro and Werner [1] at the 6th International Conference on Advances and Trends in Software Engineering (SOFTENG - IARIA), Lisboa, Portugal.

In general, software is still largely built from scratch. Ideally, existing documents (source code, project documents, etc.) could be copied and adapted to new requirements. However, the goal of reuse is still far from ideal [2]. Reuse is something intrinsic to people, and nothing is made from scratch - everything is built from something [3]. This is the basic principle of reuse, creating something from something previously built.

Software Reuse (SR) is a discipline based on this context, seeking to use existing elements (which were built to be reused) to create new ones. In general, the term SR refers to a situation in which some software is used in more than one project. In this context, software is defined as one or more items that are part of a productive organization's process. Thus, software can refer to the source code or any other product in its life cycle, such as requirements, designs, test plans, documentation, etc. [4]. With the correct use of this discipline, it can provide several positive impacts in a variety of contexts, such as quality, cost, productivity, code-making performance, rapid prototyping, reduced code writing, reliability, and interoperability of software [2].

Despite the advantages offered by reuse, discomfort is usually expressed by statements such as: "We don't want to do extra work to benefit someone else"; "We can do it better"; "We won't use it if it was created by someone else"; and even competition within organizations can be an obstacle in this regard [2]. Despite all these problems mentioned, there is still one that is seen as the main factor for reuse not being implemented, which is the difficulty of learning in this area [5].

One of the biggest obstacles that educators face in the teaching process is the need to use methods that effectively make students pay attention in class and learn easily. Many students with little class time have already lost their focus in the discipline being taught. This can be a reflection of the passive teaching method being strongly centered on the teacher, lectures, and slides without approaches that capture the student's attention [6].

Based on this, educators are increasingly looking for innovative learning strategies that combine pleasure with education so that it is possible to solve problems by teaching a subject [6]. Thus, alternative teaching methods are sought with which the student can interact and make better use of teaching, in a practical and attractive way. Based on this, a possible solution to make this learning more pleasurable is the use of games as a teaching tool. Most games are interactive, which is one of the main ways of distraction and pleasure, making them an excellent way to remind students of what has been taught [7].

The study presented in this paper is part of a more extensive study that sought to find and produce games for teaching software reuse. The study initially did not find any work that made specific reference to a game for teaching reuse. Still, it was observed that SR might be contained in different areas, such as programming, Software Engineering (SE), among others. However, the programming area is usually the most referenced [8]. Based on the information provided, this study aims to identify games that have the purpose of teaching programming with emphasis/potential for reuse, that is, to find games that were developed for teaching programming, but could be used to teach some of the fundamentals of software reuse, such as logical reasoning development, function development, object orientation, among others.

Based on each of the games found, a possible modification or its use for teaching SR was proposed. Based on these proposals, it was understood that games could be derived or reused for more than one context. With this, two more games were derived from those found for SR teaching.

The remainder of this paper is presented as follows: Section II presents some related works, Section III describes the research method used in the systematic mapping, Section IV shows some results that were found, Section V demonstrates

an example of how one of the games found could be used to teach SR, and Section VI shows the threats to validity and concludes with the final remarks.

II. Related works

This section aims to briefly present some related works that served as inspiration for this study. The papers show studies on teaching programming through games and make some observations about this method.

According to Combéfis, Beresnevičius, and Dagienė [9] several game-based platforms offer programming teaching. In their work, they sought to identify several games that aim to teach programming; each game's characteristics and elements are shown in the paper.

Malliarakis et al. [10] sought to review the features that should be supported by educational games and which educational games already support these features for programming. In this work, several games were reviewed. Their characteristics were demonstrated through a relationship table, showing each of the observed games' elements and features.

Miljanovic and Bradbury [11] evaluated 49 games for teaching programming to find characteristics of friendliness, accessibility, learning effect, and involvement. From the tables of elements presented in the paper it was possible to identify several games that sought to teach concepts involved with Software Reuse, such as functions, object orientation, and recursion. It is worth remembering that the research areas demonstrated in these papers are part of both Programming and Software Reuse, and there is an intersection of content between them.

From these works and related information, it was possible to observe that many of the observed games had essential characteristics that could be used for teaching Software Reuse.

III. Research Method

Systematic mapping is a secondary study method based on a structured and repeatable process or protocol that explores studies and provides a result in the form of an overview of a particular subject [12]. The mapping presented follows the protocol proposed by Kitchenham [13].

The research process presented in this study covers articles published by the end of 2018 and aims to conduct a systematic mapping to identify games that were built for programming teaching but could be used to teach software reuse fundamentals, such as logical reasoning development, function development, object orientation, among others.

A. Research Questions

- **Q1:** What is the main advantage / motivation of the use of games to teaching programming language?
- **Q2:** What is the disadvantage of the use of games to teaching programming language?
- **Q3:** What is the main characteristic of the game used?
- **Q4:** What was the evaluation method used?

The mapping presented followed well-defined steps so that it was possible to reach a set of articles that were of interest to the search [13]. The search string was executed in Scopus as recommended by other studies [14] [15], and then the inclusion and exclusion criteria were applied to the set of articles that were found based on the title, abstract, and full text.

B. Inclusion criteria

- The article must be in the context of using games for programming language teaching;
- The article must provide clues about software reuse
- The article must provide data to answer at least one of the research questions;
- The article should be written in English.

C. Exclusion criteria

- Book chapters, conference call;
- Studies that can not be fully accessed (i.e., papers that could not be downloaded).

D. Search string and Analysis

The definition of the search string was based on the **P**opulation, **I**ntervention, **C**omparison, **O**utcome (PICO) structure [16], using three of the four levels. The search string was defined by grouping the keywords of the same domain with the logical operator "OR" and grouping the two fields with the logical operator "AND". However, we chose to use a date filter, searching only for articles that were published within five years, aiming to find more recent works in the area [17]. Table I demonstrates the PICO structure used in conjunction with the search string. Initially, the search string returned a total of 507 papers. When analyzed according to the inclusion and exclusion filters, this number dropped to 17 papers.

According to Motta et al. [14] and Matalonga et al. [15], the snowballing procedure can mitigate the lack of other search engines, complement the strategy. Therefore, to minimize the loss of some articles, the snowballing procedure were chosen, looking at the references and citations of the articles looking for relevance [15]. The snowballing process is divided into two stages, backward and forward. The backward process aims to identify new articles based on the works that were referenced in the article that was analyzed, and the forward refers to the identification of new papers based on the works that referenced the paper that was analyzed [15]. From the use of this procedure, it was possible to include nine more papers, three of them through the forward process, and six through the backward process, resulting in a total of 26 papers. Figure 2 demonstrates a summary of the data extraction process that was used in this work. Table II shows how these 26 articles were obtained, and Table III shows each of these articles, demonstrating which questions were answered by each paper.

TABLE I. SEARCH STRING

PICO	SYNONYMS
Population	Programming language, algorithm experience, algorithm skills, algorithm alternative, algorithm method, coding experience, coding skills, coding method, coding alternative
Intervention	Tutoring, teach*,instruction, discipline, schooling, education*, mentoring, course, learn*,train*, syllabus
Comparison	Not applicable
Outcome	Game*, gami*, play*, "serious games", edutainment, "game based learning", simulation
SEARCH STRING	
TITLE-ABS-KEY (("programming language" OR "algorithm experience" OR "algorithm skills" OR "algorithm alternative" OR "algorithm method" OR "coding experience" OR "coding skills" OR "coding method" OR "coding alternative") AND (tutoring OR teach* OR instruction OR discipline OR schooling OR educat* OR mentoring OR course OR learn* OR train* OR syllabus) AND (game* OR play* OR "serious games" OR gami* OR edutainment) AND (LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014)))	

TABLE II. ANALYSIS OF THE PAPERS

	Main Study		Snowballing backward		Snowballing Forward	
Activity	Result	Number of papers	Result	Number of papers	Result	Number of papers
First Execution	507 added	507	389 added	389	123 added	123
Repeated Papers	6 withdraw	501	294 withdraw	95	16 withdraw	107
Papers in another language	0 withdraw	501	14 withdraw	81	13 withdraw	94
Remove conference / workshops	16 withdraw	485	0 withdraw	81	0 withdraw	94
Remove books	0 withdraw	485	0 withdraw	81	0 withdraw	94
Remove by title	368 withdraw	117	46 withdraw	35	58 withdraw	36
Remove by abstract	83 withdraw	34	17 withdraw	18	18 withdraw	18
Papers not found	0 withdraw	34	0 withdraw	18	0 withdraw	18
Remove by full paper	17 withdraw	17	12 withdraw	6	13 withdraw	3
Total papers included	17 papers		6 papers		3 papers	
Extracted Papers	26 papers					

TABLE III. TRACEABILITY MATRIX.

Title	Year	Q1	Q2	Q3	Q4
Perceptions of Scratch programming among secondary school students in KwaZulu-Natal, South Africa	2018	X		X	X
Robo3: A Puzzle Game to Learn Coding	2018	X	X	X	X
Improving programming skills in engineering education through problem-based game projects with Scratch	2018	X		X	X
Introducing novice programmers to functions and recursion using computer games	2018	X		X	X
Introducing programming using “scratch” and “greenfoot”	2018	X		X	X
Developing Educational 3D Games With StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	2018	X	X	X	X
Learning to think and practice computationally via a 3D simulation game	2018	X		X	X
Design and implementation of Robo3 : an applied game for teaching introductory programming	2017	X		X	X
A cross-cultural review of lightbot for introducing functions and code reuse	2017	X		X	
Using Digital Game as Compiler to Motivate C Programming Language Learning in Higher Education	2017	X		X	X
Cubely: Virtual reality block-based programming environment	2017	X		X	X
Analysis of the learning effects between text-based and visual-based beginner programming environments	2017	X		X	X
Visual programming language for model checkers based on google blockly	2017	X		X	X
Educational resource based on games for the reinforcement of engineering learning programming in mobile devices	2016	X		X	X
Teaching abstraction, function and reuse in the first class of CS1 - A lightbot experience	2016	X		X	X
From Alice to Python Introducing text-based programming in middle schools	2016	X		X	X
Visual programming languages integrated across the curriculum in elementary school: A two year case study using Scratch” in five schools	2016	X		X	X
Building a Scalable Game Engine to Teach Computer Science Languages	2015	X		X	X
A mobile-device based serious gaming approach for teaching and learning Java programming	2015	X		X	
Coding with Scratch: The design of an educational setting for Elementary pre-service teachers	2015	X		X	X
Droplet, a Blocks-based Editor for Text Code	2015	X		X	X
Integrating Droplet into Applab – Improving the usability of a blocks-based text edit	2015	X		X	X
The development of a virtual learning platform for teaching concurrent programming languages in secondary education: The use of open Sim and Scratch4OS	2014	X	X	X	X
Effects of using Alice and Scratch in an introductory programming course for corrective instruction	2014	X		X	X
A structured approach to teaching recursion using cargo-bot	2014	X		X	X
The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners, Perspective, Informatics in Education	2014	X		X	X

IV. RESULTS

The articles found in this study sought to demonstrate games that could be used in teaching some concepts related to programming. However, the analysis of the documents was performed in search of works that could be used to explain some of the concepts of SR. From this, works that were not developed with this context but could be used for this purpose were also found. Figure 2 groups the articles by location and year of publication. It is possible to see an increase in the number of publications over the years and that many countries are looking for improvements in this area.

The bottom of the figure also shows the number of articles found grouped by game type. However, some papers used more than one approach. From Figure 2, it is possible to observe that the most used way to teach programming is through the use of ”blocks of code”. Although this is not a real game, it uses many features similar to games, such as the use of graphical interfaces, sounds, and tasks to be done. Thus, these papers were also considered in this work.

Q1: What is the main advantage / motivation of the use of games to teaching programming language?

Video games have been in our lives for more than 50 years, quickly becoming one of the most important, profitable, and influential forms of entertainment [19]. A game is an activity between two or more decision-makers who seek to achieve their goals in some limited context. A more conventional definition is that games are systems in which users participate in an artificial conflict, defined by rules, which ends in a

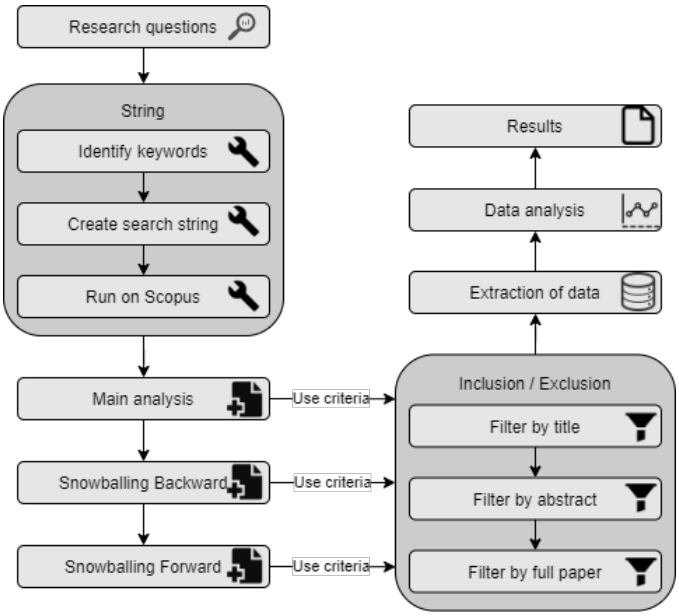

Figure 1. Summarization of the research protocol based on the model created by Calderón et al. [18]

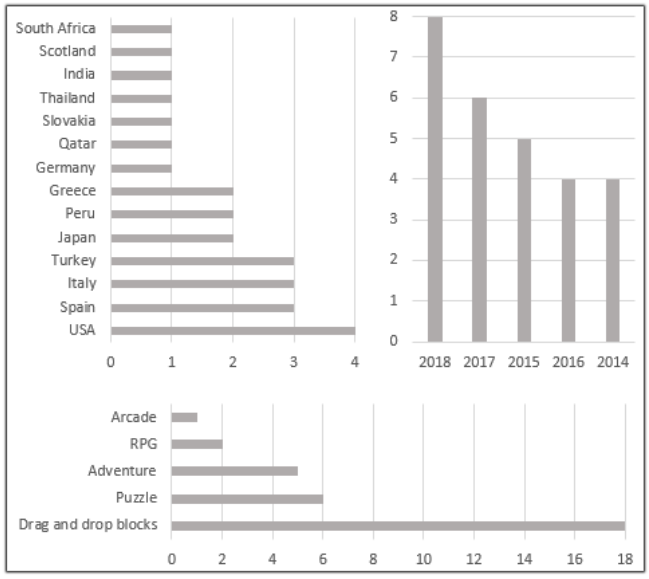

Figure 2. General analysis of the articles found.

quantifiable result [20].

Games are a visible and physical representation of a problematic space, a captured mental model that can be repeated. They are places to: test new ideas and experiment theories; repeat the training as many times as necessary; where time and space can be contracted or expanded; learn more from failure than from success [21] [19].

Many benefits are associated to the use of games in education, such as increased collaboration and competition, the creation of immediate feedback, the possibility of reflecting on the results achieved and the transfer of content so that learning is an integral part of gameplay, where the student must use the knowledge acquired in class to solve a problem within the game [22] [23] [24].

TABLE IV. Advantages of using games as a teaching technique.

Advantage	References
Entertainment	[22, 23, 27, 29–37]
Practical knowledge	[20, 23, 27, 37, 38, 38–44]
Engagement and motivation	[23, 29, 31, 35, 36]
Knowledge retention	[22, 23, 35, 36, 45]
Interactivity	[36]
Immediate feedback	[29, 31, 32, 35]
Reflection	[23, 36]
Immersion	[36]
Real Scenery	[32, 36]
Risk-free experience	[36]

Using games as a reinforcement tool to teach skills can be a very beneficial strategy for students. They have proven to be a useful tool to complement conventional learning methods. Games allow visualizing concepts that may be too abstract. They also help you get acquainted with the knowledge and methods that may be tedious to study, offering a cycle of challenges and rewards that drives the learning experience [25].

Many authors claim that games have several characteristics that can benefit teaching [22] [26]. They have already been used as successful educational tools in many different fields and topics, such as engineering, learning languages, theater and even health [27]. The advantages include: increased student motivation and engagement, enhancement of pre-existing knowledge, increased performance in practical activities, immediate feedback, fun and satisfaction, among others [25].

Finally, some visual programming languages were also identified that are not directly considered games, but that use very similar characteristics, such as increased motivation and student engagement, use of graphical interfaces, among others. Visual programming is the use of visual expressions in a programming environment as an editing shortcut to generate code that may or may not have a different syntax than that used in textual coding [28]. There are several advantages to using visual programming, and the main ones are [5] [12]: Code generation through the combination of blocks; Make programming more accessible to a specific audience; Improve the quality with which people perform programming tasks; Reduce the number of errors generated by beginning programmers, not having to worry about the language syntax; Improve the speed with which people perform programming tasks; feedback and visual execution; and Minimize the command set.

Several advantages were found in relation to the use of games as a teaching strategy, Table IV demonstrates each one of these.

Q2: What is the disadvantage of the use of games to teaching programming language?

The limited use of serious games in formal education may be related to the issues surrounding the use of leisure games, such as a view where games can be addictive, are not productive, and can teach wrong concepts. Another very relevant point regarding the non-use of games as a teaching medium is that players usually learn through repetition, patterns, and exploration, which contrasts with the learning of

discrete amounts of information, as it occurs in schools and undergraduate courses in general [46].

Despite the advantages offered by games as a teaching method, there are also some issues involving this approach. The first problem found was the comparison of the level of learning provided by a game as a teaching method and a class with textual programming. Despite the advantages offered by games, textual programming can still convey better content [35]. Despite the advantages offered by visual programming, it was also observed that text-based programming achieved better results in relation to knowledge transfer [21].

Another problem identified was the complexity of the game created. If the teaching tool used is too complicated, students can reduce the time spent solving problems to focus more on the tool. This is an unwanted distraction, and any game used should be easy to use, allowing the student to focus on solving the problem rather than how to use the game [38].

Finally, the last problem identified was that although games provide several advantages, they are not seen as self-sufficient. Professional follow-up and feedback on the course are required to solve any problem that may arise throughout the learning process [38]. Therefore, it is unlikely to teach students new content just by using games. They are mostly used to reinforce content that has already been presented by a teacher.

Q3: What is the main characteristic of the game used?

This study identified several games that sought to teach programming through increased motivation and engagement through fun. Most of these games were designed to be used by users with minimal or no knowledge of programming languages [45].

The first game found was LightBot, which is a game to teach programming logic and has features such as multi-level, difficulty progression, feedback, challenges, use of similar tasks, concepts of functions, abstraction, flow control and recursion [29] [31] [34]. Another game very similar to the one described above is Cargo-Bot, which has the same characteristics, but with other gameplay that revolves around a crane that moves and stacks a set of colored boxes. Players write small programs to move boxes from one initial setup to another [23]. Another game called Robo3 was found that had characteristics very similar to those described [25].

Another game very similar to the ones listed above was a game designed to teach Java programming that, to advance the levels, the player needs to overcome different levels. As the player surpasses these levels, he or she can progress through the story, unlocking new elements and gaining experience points to unlock new content [45].

Another game found was Lost in Space, which includes, among other components, a game rules system, a physics engine, and a rendering engine. The game screen is divided into two parts. The left side containing the code interpreter text area and a help window and on the other side, the game phase. Through this game, some features were highlighted, such as obstacles, code interpreter (pseudocode of the game), collisions, movement, enemy and attack system [27].

In this research, we also identified some visual programming languages that are not considered as games directly, but that uses ”block” approach to building programs. The first two to be identified were Alice [32] and Scratch [30], which are block-based visual programming languages designed to promote media manipulation for new programmers. From these languages, it is possible to upload media projects and scripts, animated stories, games, book reports, greeting cards, music videos, tutorials, simulations, and art and music projects. Two other languages very similar to those described are StarLogo TNG [35] and Droplet [47] [48], which are also drag-and-drop visual languages.

Greenfoot is an integrated tool that aims to teach object-oriented programming. Also, the tool allows teachers to introduce the most essential and fundamental concepts of object orientation in an easily understandable way [49]. Finally, the last visual language found is called Google Blockly [43], which is a library for building visual programming editors.

Finally, another feature that was used to create these games was the use of virtual and augmented reality. The Cubely game made use of these technologies to develop an idea that blended block programming concepts and the Minecraft game [36].

Several characteristics were found in the games that were evaluated. Table V shows each of them.

TABLE V. Characteristics used in games as teaching techniques.

Characteristics	References
Score	[17, 23, 25, 27, 29, 31, 45, 50]
Levels	[23, 25, 27, 29, 31, 45]
Checkpoints	[23, 25, 29, 31, 45]
Competition and collaboration	[22]
Feedback	[23, 25, 27, 29, 31]
Simulation	[36]
Final result	[23, 27, 29, 31, 36, 45]
Real scenario	[36]
Challenge	[22, 23, 25, 27, 29, 31, 36, 45]
Dependence between contents	[22, 29]
Stimulating graphics	[22, 23, 27, 29, 36]

Q4: What was the evaluation method used?

Several evaluation methods were identified in this research. However, in general, all evaluations have a questionnaire applied to a specific population after using the tool to validate it [35] [32] [43].

Another possible means of the evaluation was the use of control groups where one group used the tool, and the other did not, and the same questionnaire was applied to both groups [27]. Through this assessment, it is possible to find out if there was a gain of experience through the tool use since it is possible to compare the results of the two groups.

The last evaluation method found was about the use of the tool as part of the discipline — the tool as a complement to the teaching of programming [42].

V. Discussion

A. Reusing games to teach SR

This mapping found several games; however, none of them was produced to teach SR. Nevertheless, these games, with only a few or no modifications, could be used to explain certain concepts of SR, such as the importance of reusing, software components or code reuse.

Thinking about this idea of teaching SR, visual language platforms, such as Scratch [30], Alice [32], Droplet [47], and Google Blockly [43] could, for example, be used to teach code reuse or software components. A software component can be understood as a software unit with a well-defined interface and explicitly specified dependencies. It can have different sizes and can be characterized in different ways, from a small piece of code, a software package, a web service, a module that encapsulates a set of functions, or it can even be as large as a program [2]. All the visual languages have a strong base of reuse, where the user creates programs by joining blocks (i.e., pre-produced components). Therefore, it is possible to understand the "blocks" of visual programming code as software components. Both (blocks and components) can be seen as pieces of code that exercise a specific functionality and must be reused to produce a new program.

Robo3 [25], CargoBot [23] and Lightbot [29] are games of puzzle type and are very similar. The general idea of these games is to create sequences of activities (which are described as functions) that perform a task, such as taking the avatar from point A to point B or moving boxes from one initial setup to another. Thinking about this type of games, these functions can be used in the game several times, teaching the student the concept and importance of code reuse. Cubely [36] is a game based on Minecraft, and its mechanics can be understood as combining a puzzle game with a bit of visual programming. In this game, the user must create cubes with a pre-defined action that looks like a software component. With this in mind, it is possible to use the game to teach components and the importance of reusing software, forcing the user to reuse to complete the puzzle.

Lost in Space [27] was another game found that could be used to teach software components. A tittle modification in the game mechanics would be enough, the idea would be to use components to control the game ship or change the dynamics of the game and make the player have to create a component that aims to destroy the ship.

Finally, a similar puzzle game was also found to teach Java programming [45]. This game has a collection of activities divided into levels where the user needs to write code snippets, taking the character from point A to point B to avoid obstacles. This game with minor modifications could also be used to teach software components, where at each phase, the player would need to create a component to reuse throughout the other levels.

B. Derivation games

Most games are built from derivations of others with minor modifications to sprites, stories, or mechanics. The reason so many games look similar is that they use the same set of mechanics with small changes [51].

When discussing modifications and derivations of games, a well-known term in this area is mod. This term can be defined as any form of non-commercial change of a proprietary digital game [52]. Through this term, it is possible to notice that many games are built from others, for example, when entering the website moddb.com, it is possible to find more than 21000 modified projects (accessed on June 28, 2020). There are other terms to define games that were created from others. This characterization depends on the size of the modification that was made. Table VII demonstrates each of the categorizations and presents a discussion of whether such a modification is a derivation or not [52] [53].

Games are formed by different elements that combined give life to the experience lived by the player. Small changes in these elements make it possible to derive a new game that will provide new experiences. Different authors divide these elements in different ways, and in the following three ways to divide these elements will be demonstrated.

The first method of dividing game elements is MDA. This method is divided into three essential elements: mechanics, dynamics, and aesthetics. **Mechanics** can be understood as the rules and actions that can happen during the game. **Dynamics** represent the behavior that occurs as a result of actions. Finally, **aesthetics** are the emotions experienced by the player [54].

The second method of dividing the elements of games that was found was the proposal made by Jesse Schell [51]. In this method, every game consists of four essential elements: aesthetics, mechanics, history, and technology. **Aesthetics**, in this case, can be understood as what gives the appearance to the game, anything that interacts with the player, such as images and sounds. **Mechanics** is the rules and actions that can happen; **history** is the narrative of the game, the events that are happening throughout the game. Finally, **technology** is the junction point of all others; it is where the game is built [51].

The last method found was that proposed by Aki Jarvinen [55]. This method divides the elements into some groups: **mechanics** that have already been defined; **rules** that are the procedures with which the game system restricts and moderates the game; **theme** that is the subject of the game, the plot; and **information** that can be understood as what the system stores and presents to the player.

Based on these elements that make up the games, it is possible to create derivations of existing games by making small changes to parts of these elements. For example, think of a simple card game like canasta. This game has several variations that use the same technology (the cards), mechanics, aesthetics and dynamics similar to just a few changes in the rules. However, it is important to note that it is not any change in an element that generates a new game. For example, a change in technology or appearance (sprites, sounds) in the game will not create a new game. To try to make it more transparent to the reader what can be modified to derive one game from another, Table VII groups the similar elements and presents a brief discussion about each one.

Based on the idea of derivation, two games were built using the main mechanics and characteristics found in this mapping. The idea was to create games derived from those found to teach SR. The first game was a simple quiz to demonstrate the information to be taught in an SR discipline in a way that was not only through the reading of books and slides. The quiz used characteristics such as response time, demonstration of the right and wrong answers as feedback, progress bar, and difficulty levels. Thinking about derivation, this game could be easily modified and applied in other contexts different from teaching of SR. It would be enough to modify the questions in the game to satisfy the criteria of another discipline. The second game explored the derivation idea further and was inspired by the Lightbot, using a similar mechanic with some modifications. The game created was called CodeBoy and can

TABLE VI. CATEGORIZATION OF CHANGES, ADAPTED FROM [52, 53]

Name	Description	Discussion
Mutators or tweaks	Minor modification, such as changing the speed of the game. Mutators can also be additions that do not influence the game and its mechanism, having only an aesthetic effect.	It is not considered a derivation because it has only aesthetic effects, or very small modifications.
Add-ons	Provides some extensions, such as new maps, new units, and new skins, among others.	It is not considered a derivation because it is just an extension.
Mods	Can include add-ons and mutators, or rather, they also manipulate the rules system and the visual layer. Often, they try to establish a new version of the game by modifying much of it.	Alter the original game significantly, and therefore considered a derivation.
Total conversions	Manipulate the original game in so many ways that a new game is created. Still, these changes can be applied to different levels of the game.	

TABLE VII. DISCUSSION OF ELEMENTS.

Element	Discussion
Mechanic (Actions)	Modifications to the actions that the user can perform in a game will create a derivation. For example, think in a chess where the pieces have different movements.
Mechanic (Rules)	Changes in the rules that the user can execute in a game will create a derivation. For example, think of a target shooting game where the player must play blindfolded, the game will be more difficult through this new rule added.
Dynamic	Dynamics represents the behavior that occurs as a result of actions. To have a change in the dynamics of the game, you must change part of the mechanics. Therefore, an isolated change in this element would not occur.
Aesthetic	Aesthetics are the emotions experienced by the player. It will hardly be possible to make an isolated change in this element. It will be necessary to modify another element to influence it.
Aesthetic (Appearance) Information	Modifications in the appearances (sprites, sounds) of games will not create a new game. Playing Zelda without music is still the same game.
Story/Theme	Changes in stories or themes of games will result in a new game. Many developers use this approach, creating new games that only change the story. For example, there are several action games with walking and shooting mechanics but with different stories.
Technology	The change in technology does not create a new game. It can change the experiences experienced by the player. However, it remains the same game. Computer chess remains a game of chess.

be used to demonstrate the importance of SR, showing the student that there are certain moments when it is essential and often indispensable. CodeBoy used movement mechanics similar to Lightbot as jumping, rotating, and walking. This game is also integrated with the idea of the FODA (Feature-oriented domain analysis) language tree in the context of SR to permit the construction of functions [56].

The Codeboy derivation process followed three phases, namely: analysis, division, and derivation. The analysis seeks to obtain an understanding of the game to be derived, collecting information such as the objective of the game and how it works. The division aims to separate the game according to the elements of the Table VII, to understand each one separately. Finally, in the derivation phase, each element is reviewed to understand which elements will be modified or reused. Table VIII shows the division phase with each of the elements of the two games. It is worth remembering that when an element was reused, its discussion will be demonstrated only once. Figure 3 demonstrates Lightbot [29] at the top and CodeBoy [56] at the bottom. A better understanding of the games created and their evaluations can be found in Castro and Werner [56].

Figure 3. Lightbot [56] and Codeboy [29]

VI. FINAL REMARKS

A. Threats to Validity

Through a critical analysis of the mapping, it is possible to perceive some threats that may have affected the final result of the work. The first to be highlighted is about the period in which the mapping was performed, collecting information from just five years. The second threat is the problem of interpreting the information found, which is up to the author to understand the game found and think of a way that could be applied in the teaching of SR.

TABLE VIII. LIGHTBOT VS CODEBOY.

Element	LightBot	CodeBoy
Objective	Teach programming logic through sequence of movements as an algorithm	Teach reuse through the development of software components
Mechanic (Actions)	1 - Drag and drop blocks to move the robot 2 - Create functions with a sequence of steps for character movement 3 - Turn on the lights along the way	1 - Drag and drop blocks of codes to move the boy and solve the phase algorithm 2 - Create functions with a sequence of steps for character movement 3 - Catch the gold star 4 - Open the magic chest
Mechanic (Rules)	Create the sequence of correct movements to move the robot	The basis of the game is through the construction of components. It is worth remembering that these created components can be saved and reused in later stages
Dynamic	Watch the robot move and turn on the lights along the route	After creating the algorithm, it will be validated through metrics and a score will be calculated for the player, encouraging the user to create better programs to achieve higher grades
Aesthetic	The user must think about the correct movement sequence (challenge)	The user must develop his own component (expression) based on the algorithm described in the phase, so that he can get an adequate grade to advance to the next phase (challenge)
Aesthetic (Appearance) Information	1 - Maps divided into squares where the game character moves around the squares. 2 - Group of commands to be executed positioned on the right side of the screen.	1 - Maps divided into squares where the game character moves around the squares. 2 - Group of commands to be executed positioned on the right side of the screen, within the tree.
Story/Theme	Help the robot to move from point A to point B by raising the lights along the way.	Help the character move from the start to capture the gold star and then open the treasure chest.
Technology	Mobile	Mobile e desktop

B. Conclusion and Future Work

For many people who are not directly linked to the software reuse area, they refer to it as just code. Due to this fact, this mapping sought to find programming teaching games that could be used to teach reuse concepts that are often abstract to many students. From this, it was possible to identify six games and six block-based programming languages. The game, and the visual programming language that were identified in more articles were LightBot [29] and Scratch [30], respectively. The main characteristics found were the use of rules, phases, difficult progression, feedback, challenges, and the use of similar tasks in sequence.

As mentioned before, software reuse is inserted in several contexts, and the most common are propagation and engineering. This work sought to identify games that were created to teach programming but could be used to explain some of the fundamentals of software reuse, thus looking at works from the first context. To better understand how these games are used as teaching methods, it is intended to perform another mapping to identify games that aim to teach software engineering, since as software reuse is inserted in the engineering and possibly similar features can be used to the teaching of the two subjects.

Although this work has found some games that could be used to teach some reuse fundamentals such as components, functions, and object orientation, none of these games were specifically designed to teach software reuse. Based on each of the games found, a possible modification or use of it for teaching SR was proposed. Based on these proposals, it was understood that games could be derived or reused for more than one context. Most games are created from others with minor modifications, this process is called derivation. In order to test this process, a three-step protocol was built and used to create a game from it.

Games can be a new method to complement the current teaching method due to its main advantages, such as increased practice and engagement through challenges, rewards, fun, and feedback. However, it is still something new that needs attention due to problems such as the complexity of the game that can affect learning, and the level of learning provided by games is still lower than current teaching methods.

As mentioned before, several block-based programming languages have been found, showing that this strategy has also been used on a large scale for teaching programming. Based on this, it is intended to create games that use this strategy to help teaching Software Reuse.

To conclude, it is expected to improve and systematize the game evolution showed in this paper to continue the derivation of games more transparently and efficiently.

REFERENCES

[1] D. Castro and C. Werner, “Mapping on the use of games for programming teaching with an emphasis on software reuse,” The Sixth International Conference on Advances and Trends in Software Engineering, SOFTENG, February, 2020.

[2] J. Sametinger, Software engineering with reusable components. Springer Science & Business Media, 1997.

[3] O. Serrat, “Harnessing creativity and innovation in the workplace,” in Knowledge solutions. Springer, 2017, pp. 903–910.

[4] R. J. Leach, Software Reuse: methods, models, and costs. McGraw-Hill New York, 1997.

[5] N. Niu, D. Reese, K. Xie, and C. Smith, "Reuse a" software reuse" course," in American Society for Engineering Education. American Society for Engineering Education, 2011.

[6] U. Ritterfeld, M. Cody, and P. Vorderer, Serious games: Mechanisms and effects. Routledge, 2009.

[7] H. Ludens, "A study of the play element in culture," Trans. by RFC Hull.(London, 1949), vol. 168, 1955.

[8] T. Mikkonen and A. Taivalsaari, "Software reuse in the era of opportunistic design," IEEE Software, vol. 36, no. 3, 2019, pp. 105–111.

[9] S. Combéfis, G. Beresnevičius, and V. Dagienė, "Learning programming through games and contests: overview, characterisation and discussion," Olympiads in Informatics, vol. 10, no. 1, 2016, pp. 39–60.

[10] C. Malliarakis, M. Satratzemi, and S. Xinogalos, "Educational games for teaching computer programming," in Research on e-Learning and ICT in Education. Springer, 2014, pp. 87–98.

[11] M. A. Miljanovic and J. S. Bradbury, "A review of serious games for programming," in Joint International Conference on Serious Games. Springer, 2018, pp. 204–216.

[12] I. Steinmacher, A. P. Chaves, and M. A. Gerosa, "Awareness support in distributed software development: A systematic review and mapping of the literature," Computer Supported Cooperative Work (CSCW), vol. 22, no. 2-3, 2013, pp. 113–158.

[13] B. Kitchenham, "Procedures for performing systematic reviews," Keele, UK, Keele University, vol. 33, no. 2004, 2004, pp. 1–26.

[14] R. C. Motta, K. M. de Oliveira, and G. H. Travassos, "Characterizing interoperability in context-aware software systems," in 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE, 2016, pp. 203–208.

[15] S. Matalonga, F. Rodrigues, and G. H. Travassos, "Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review," Journal of Systems and Software, vol. 131, 2017, pp. 1–21.

[16] M. Petticrew and H. Roberts, Systematic reviews in the social sciences: A practical guide. John Wiley & Sons, 2008.

[17] S. Jiang, H. Zhang, C. Gao, D. Shao, and G. Rong, "Process simulation for software engineering education," in Proceedings of the 2015 International Conference on Software and System Process. ACM, 2015, pp. 147–156.

[18] A. Calderón, M. Trinidad, M. Ruiz, and R. V. O'Connor, "Teaching software processes and standards: A review of serious games approaches," in International Conference on Software Process Improvement and Capability Determination. Springer, 2018, pp. 154–166.

[19] S. Ramírez-Rosales, S. Vázquez-Reyes, J. L. Villa-Cisneros, and M. De León-Sigg, "A serious game to promote object oriented programming and software engineering basic concepts learning," in 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE, 2016, pp. 97–103.

[20] K. Salen, K. S. Tekinbaş, and E. Zimmerman, Rules of play: Game design fundamentals. MIT press, 2004.

[21] C. Caulfield, J. C. Xia, D. Veal, and S. Maj, "A systematic survey of games used for software engineering education," Modern Applied Science, vol. 5, no. 6, 2011, pp. 28–43.

[22] T. Jordine, Y. Liang, and E. Ihler, "A mobile-device based serious gaming approach for teaching and learning java programming," in 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, 2014, pp. 1–5.

[23] E. Lee, V. Shan, B. Beth, and C. Lin, "A structured approach to teaching recursion using cargo-bot," in Proceedings of the tenth annual conference on International computing education research. ACM, 2014, pp. 59–66.

[24] R. O. Chaves, C. G. von Wangenheim, J. C. C. Furtado, S. R. B. Oliveira, A. Santos, and E. L. Favero, "Experimental evaluation of a serious game for teaching software process modeling," ieee Transactions on Education, vol. 58, no. 4, 2015, pp. 289–296.

[25] F. Agalbato, "Design and implementation of robo3: an applied game for teaching introductory programming," Scuola di Ingegneria Industriale e dell'Informazione, 2017.

[26] R. Atal and A. Sureka, "Anukarna: A software engineering simulation game for teaching practical decision making in peer code review." in QuASoQ/WAWSE/CMCE@ APSEC, 2015, pp. 63–70.

[27] Á. Serrano-Laguna, J. Torrente, B. M. Iglesias, and B. Fernández-Manjón, "Building a scalable game engine to teach computer science languages," IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, vol. 10, no. 4, 2015, pp. 253–261.

[28] M. M. Burnett, "Visual programming," Wiley Encyclopedia of Electrical and Electronics Engineering, 2001.

[29] E. V. Duarte and J. L. Pearce, "A cross-cultural review of lightbot for introducing functions and code reuse," Journal of Computing Sciences in Colleges, vol. 33, no. 2, 2017, pp. 100–105.

[30] D. Topalli and N. E. Cagiltay, "Improving programming skills in engineering education through problem-based game projects with scratch," Computers & Education, vol. 120, 2018, pp. 64–74.

[31] R. Law, "Introducing novice programmers to functions and recursion using computer games," in European Conference on Games Based Learning. Academic Conferences International Limited, 2018, pp. 325–334.

[32] C.-K. Chang, "Effects of using alice and scratch in an introductory programming course for corrective instruction," Journal of Educational Computing Research, vol. 51, no. 2, 2014, pp. 185–204.

[33] N. Pellas and S. Vosinakis, "Learning to think and practice computationally via a 3d simulation game," in Interactive Mobile Communication, Technologies and Learning. Springer, 2017, pp. 550–562.

[34] M. Aedo Lopez, E. Vidal Duarte, E. Castro Gutierrez, and A. Paz Valderrama, "Teaching abstraction, function and reuse in the first class of cs1: A lightbot experience," in Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. ACM, 2016, pp. 256–257.

[35] N. Boldbaatar and E. Şendurur, "Developing educational 3d games with starlogo: The role of backwards fading

in the transfer of programming experience," Journal of Educational Computing Research, vol. 57, no. 6, 2019, pp. 1468–1494.

[36] J. Vincur, M. Konopka, J. Tvarozek, M. Hoang, and P. Navrat, "Cubely: Virtual reality block-based programming environment," in Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology, ser. VRST '17. New York, NY, USA: ACM, 2017, pp. 84:1–84:2. [Online]. Available: http://doi.acm.org/10.1145/3139131.3141785

[37] M. Marimuthu and P. Govender, "Perceptions of scratch programming among secondary school students in kwazulu-natal, south africa," African Journal of Information and Communication, vol. 21, 2018, pp. 51–80.

[38] N. Pellas, "The development of a virtual learning platform for teaching concurrent programming languages in the secondary education: The use of open sim and scratch4os," Journal of e-Learning and Knowledge Society, vol. 10, no. 1, 2014, pp. 129–143.

[39] H. Pötter, M. Schots, L. Duboc, and V. Werneck, "Inspectorx: A game for software inspection training and learning," in 2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2014, pp. 55–64.

[40] N. Tabet, H. Gedawy, H. Alshikhabobakr, and S. Razak, "From alice to python. introducing text-based programming in middle schools," in Proceedings of the 2016 ACM Conference on innovation and Technology in Computer Science Education, 2016, pp. 124–129.

[41] F. Kalelioglu and Y. Gülbahar, "The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective." Informatics in Education, vol. 13, no. 1, 2014, pp. 33–50.

[42] L. A. Vaca-Cárdenas, F. Bertacchini, A. Tavernise, L. Gabriele, A. Valenti, D. E. Olmedo, P. Pantano, and E. Bilotta, "Coding with scratch: The design of an educational setting for elementary pre-service teachers," in 2015 International Conference on Interactive Collaborative Learning (ICL). IEEE, 2015, pp. 1171–1177.

[43] S. Yamashita, M. Tsunoda, and T. Yokogawa, "Visual programming language for model checkers based on google blockly," in International Conference on Product-Focused Software Process Improvement. Springer, 2017, pp. 597–601.

[44] J.-M. Sáez-López, M. Román-González, and E. Vázquez-Cano, "Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools," Computers & Education, vol. 97, 2016, pp. 129–141.

[45] A. Sierra, T. Ariza, F. Fernández-Jiménez, J. Muñoz-Calle, A. Molina, and Á. Martín-Rodríguez, "Educational resource based on games for the reinforcement of engineering learning programming in mobile devices," in 2016 Technologies Applied to Electronics Teaching (TAEE). IEEE, 2016, pp. 1–6.

[46] D. Valencia, A. Vizcaino, L. Garcia-Mundo, M. Piattini, and J. P. Soto, "Gsdgame: A serious game for the acquisition of the competencies needed in gsd," in 2016 IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW). IEEE, 2016, pp. 19–24.

[47] D. Bau, "Droplet, a blocks-based editor for text code," Journal of Computing Sciences in Colleges, vol. 30, no. 6, 2015, pp. 138–144.

[48] D. A. Bau, "Integrating droplet into applab—improving the usability of a blocks-based text editor," in 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). IEEE, 2015, pp. 55–57.

[49] H. Chandrashekar, A. G. Kiran, B. Uma, and P. Sunita, "Introducing programming using "scratch" and "greenfoot"," Journal of Engineering Education Transformations, 2018.

[50] R. d. A. Mauricio, L. Veado, R. T. Moreira, E. Figueiredo, and H. Costa, "A systematic mapping study on game-related methods for software engineering education," Information and software technology, vol. 95, 2018, pp. 201–218.

[51] J. Schell, The Art of Game Design: A book of lenses. CRC press, 2008.

[52] A. Unger, "Modding as part of game culture," in Computer Games and New Media Cultures. Springer, 2012, pp. 509–523.

[53] O. Sotamaa, "When the game is not enough: Motivations and practices among computer game modding culture," Games and Culture, vol. 5, no. 3, 2010, pp. 239–255.

[54] R. Hunicke, M. LeBlanc, and R. Zubek, "Mda: A formal approach to game design and game research," in Proceedings of the AAAI Workshop on Challenges in Game AI, vol. 4, no. 1, 2004.

[55] A. Järvinen, "Introducing applied ludology: Hands-on methods for game studies." in DiGRA Conference. Citeseer, 2007.

[56] D. Castro and C. Werner, "Use of games as a strategy for teaching software reuse," SBGAMES, 2020 (in Portuguese).