SOLID Tanks: A Serious Game Proposal to Teach SOLID Design Principles

Miguel Nascimento
COPPE/Computer Systems Engineering Program, Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
miguelsan@cos.ufrj.br

Diego Castro
COPPE/Computer Systems Engineering Program, Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
diegocbcastro@cos.ufrj.br

Cláudia Werner
COPPE/Computer Systems Engineering Program, Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
werner@cos.ufrj.br

Abstract

With market demands for high-quality software, students need to develop the necessary skills and knowledge to be better prepared for industry demands. Serious games (SGs) provide a risk-free and engaging environment that can better motivate students to put theoretical concepts into practice. Thus, such games can be used to assist the learning of good Software Engineering practices. This paper presents a proposal for a SG aimed at teaching the SOLID principles of software design. The purpose of this game is to introduce students to these principles through an engaging experience and improve the learning process.

Keywords

Software Engineering, Software Design, Serious Games, Game-Based Learning, SOLID principles

1 Introduction

Software developers require Software Engineering (SE) skills to produce reliable, maintainable, and extensible software that is demanded by the market. However, undergraduates and recent graduates often lack the skills expected by the industry [6, 14], which can limit their employment opportunities.

This gap may stem from the predominantly theoretical nature of many SE courses, where concepts are introduced through lectures and later applied in course projects. Given the practical nature of SE, this approach can reduce engagement and hinder the application of knowledge to real-world problems [6, 8]. Moreover, the complexity of advanced software design topics may limit their coverage in undergraduate curricula [11], leaving students with few opportunities to study and practice concepts such as the SOLID principles.

The SOLID design principles [12] promote scalable, maintainable, and reusable object-oriented software [3]. Learning these principles can help bridge the gap between students and industry practice.

This paper presents a proposal for a serious game aimed at introducing undergraduate students to the SOLID principles through code refactoring activities. The game seeks to complement undergraduate SE courses by providing students with an engaging first practical experience with software design principles.

2 Theoretical Background

2.1 Serious Games

Serious Games (SGs) refer to games that are designed with a purpose other than entertainment [6]. In education, they provide engaging and risk-free environments where students can experiment, learn from mistakes, and receive immediate feedback, increasing motivation, engagement, and learning outcomes [6, 8].

In fact, a Systematic Literature Review (SLR) conducted by Kosa et al. reported mostly positive outcomes of game-based approaches in SE education [9]. Additionally, another SLR by Silva et al. revealed that 97% of the studies reviewed report that using games is an effective way to teach programming concepts [17].

For these reasons, SGs are particularly suitable for Software Engineering education, where students benefit from repeated practice in realistic but risk-free scenarios. By combining instructional objectives with interactive experiences, SGs can encourage experimentation while reducing the fear of making mistakes.

2.2 Game-Based Learning

Game-Based Learning (GBL) is a methodology that integrates educational content into games, providing immersive environments that increase student engagement, promote active learning, and can lead to more meaningful learning experiences than traditional approaches [20].

By combining educational objectives with game mechanics, GBL encourages students to actively participate in the learning process while receiving immediate feedback in a motivating environment. In addition to increasing engagement, GBL can improve learning outcomes, academic performance, and the development of skills such as problem solving and decision making [20].

Rather than presenting theoretical concepts through passive instruction, GBL encourages students to construct knowledge by solving problems, receiving immediate feedback, and reflecting on the consequences of their actions during gameplay.

2.3 SOLID Principles

The SOLID principles [12] are five object-oriented design principles intended to improve the modularity, maintainability, and extensibility of software systems. The five principles are described below:

- **Single Responsibility Principle (SRP):** A class should have only one reason to change.
- **Open/Closed Principle (OCP):** Software should be open for extension but closed for modification.
- **Liskov Substitution Principle (LSP):** Subclasses should preserve the expected behavior of their base classes.
- **Interface Segregation Principle (ISP):** Interfaces should be smaller and more specific, avoiding unnecessary dependencies.
- **Dependency Inversion Principle (DIP):** Code should depend on abstractions rather than concrete implementations.

According to Balim et al., empirical studies indicate that their application reduces code smells while improving cohesion and modularity, contributing to more sustainable systems; however, these principles are still applied inconsistently in practice [3].

3 Related Works

The integration of SGs in SE education has been widely investigated as an alternative approach to support the learning process and increase student engagement, thereby improving students' skills and knowledge. Castro et al. [6] conducted a tertiary mapping on how SGs are used for teaching SE. The mapping shows that software design is among the least explored areas, revealing an unexplored research field in the application of SGs.

This finding is further supported by Furtado et al. [7], who conducted a Systematic Mapping about the use of SGs in the teaching-learning process in SE. One of the research questions of their study pertains to the context of SE in which SGs are applied, and none of the reported SGs focus on the area of software design. Although the most recent mapping was published two years ago, it still highlights a lack of educational games focused on software design concepts.

Nevertheless, some studies have addressed advanced software design concepts. Refactal [5] teaches software design through an integrated refactoring environment in which students identify code smells, reason about software quality using SOLID principles and GRASP patterns, and apply design patterns to improve existing code. The game emphasizes the interdependence of these topics during refactoring tasks, encouraging learners to analyze design trade-offs and justify their decisions.

Vahldick and Nolli [19] developed Relics of Chaos, a Serious Role-Playing Game for introducing the Visitor design pattern. Players complete instructional tasks involving class diagrams, source code analysis, and identification of pattern elements in code fragments. The game emphasizes initial understanding of the pattern through activities aligned with lower cognitive levels of Bloom's taxonomy before requiring students to apply their knowledge.

Similarly, Alvarez and Cieza-Mostacero [1] developed a SG for learning software design patterns through interactive exploration and assessment activities. Players navigate three environments dedicated to creational, structural, and behavioral patterns, accessing instructional materials and completing quizzes. The game combines pattern descriptions, applicability information, class diagram examples, and quizzes to support the learning of design patterns.

To the best of the authors' knowledge, no serious game has been proposed with the primary objective of teaching the SOLID principles. While Refactal [5] includes SOLID, it adopts a broader approach to software design that also addresses GRASP patterns and code smells. In contrast, SOLID Tanks focuses specifically on SOLID through code-refactoring challenges integrated into a programming-and-combat context. Furthermore, SOLID Tanks combines two automated assessment strategies: normalized AST comparison for structurally constrained refactorings and SonarQube-based code-quality analysis for challenges that may allow multiple valid implementations. This combination of SOLID-focused refactoring, gameplay integration, and automated assessment distinguishes SOLID Tanks from existing SGs for software engineering education.

4 Game Design

The game was designed using MEDIEVAL (Method for Designing Virtual Educational Instruments with a Playful Approach) [15], a 10-step framework to guide the development of serious games. As part of this process, the Revised Bloom's Taxonomy [10] was used to define learning objectives, Endo-GDC [18] to organize the educational game concept, and the Short Game Design Document (SGDD) [13] to document planned features and design specifications. Figure 1 summarizes the MEDIEVAL steps and the project's current development stage.

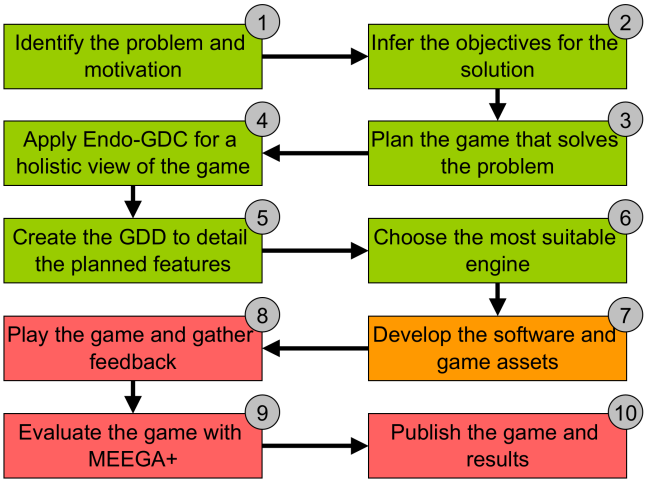

Figure 1: The ten MEDIEVAL steps. Green, orange, and red indicate completed, ongoing, and pending steps, respectively.

The learning objectives were defined at the Understand and Apply levels of the Revised Bloom's Taxonomy. The first is to understand the benefits of SOLID by interpreting the consequences of code modifications, while the second is to apply SOLID principles through code modifications. In the Endo-GDC, these objectives are associated with the game's pedagogical content: the SOLID principles and the importance of code flexibility. Thus, students first examine code that violates a target principle and its consequences before addressing the design problem through refactoring.

In SOLID Tanks, players assume the role of an engineer responsible for defending a lunar base from alien attacks. The space-themed setting was partly inspired by the arcade game Asteroids [2], particularly its concept of flying saucers appearing from outside the screen, and adapted to provide context for the tank engineer role. Designed for students with prior knowledge of object-oriented programming, the game is also inspired by Robocode [4]. SOLID Tanks adopts its programming-and-combat premise, but makes software-design refactoring the central educational activity rather than tank programming.

Each level focuses on a specific SOLID principle. Players refactor C# source code that violates the target principle while preserving its expected behavior. Once the solution satisfies the assessment criteria, the player advances to a combat scene in which the tank defends the lunar base. Some levels may also include optional refactoring challenges involving multiple SOLID principles. These challenges

are not required to unlock combat and reward players with power-ups, such as shields or support drones, connecting programming performance with gameplay rewards.

The game supports two alternative assessment strategies, with only one assigned to each challenge. The first uses Roslyn, Microsoft's .NET compiler platform, to compare normalized Abstract Syntax Trees (ASTs) from the student's solution and a reference implementation. The comparison retains syntax node and token types while normalizing identifier names and string literals, allowing variations in class, method, variable, and parameter names. Class signatures are sorted, making declaration order irrelevant. This strategy assesses structural rather than textual or semantic equivalence and is used for refactorings with a well-defined structure, such as reorganizing responsibilities for the SRP.

The second submits the student's source code to an ASP.NET backend, which executes SonarScanner and retrieves SonarQube static analysis metrics. ASP.NET provides a C#-based backend that integrates the external analysis pipeline while separating code analysis from the game client. For open-ended challenges, configurable metrics such as cognitive complexity, number of classes and functions, and code smells are combined into a weighted heuristic score, with weights adjusted per challenge. The resulting score determines whether the solution satisfies the challenge's quality criteria. This strategy is more suitable for complex challenges involving multiple SOLID principles and structurally different valid implementations.

Because the SonarQube-based strategy evaluates source-code quality rather than behavior, it does not guarantee functional correctness. It currently verifies whether open-ended refactorings satisfy the configured criteria, while behavioral testing remains a limitation.

Unity was selected for its 2D game development support and native C# environment, facilitating Roslyn integration. Figure 2 shows the programming and combat scenes. The visual indicator in the programming scene informs whether the current code satisfies the assessment criteria. The game remains under development, with software implementation and asset production in progress.

At the current stage, the AST-based comparison and backend SonarScanner analysis are implemented, together with the basic tank and alien behaviors, user interface, code editor, and challenges for SRP and OCP. Remaining work includes the other three SOLID principles, optional challenges, level-specific combat, and refinement of assessment metrics, tank and alien behaviors, and user interface.

The remaining MEDIEVAL steps include playtesting and formal evaluation using MEEGA+ [16]. The evaluation will assess aspects of player experience and usability and investigate the extent to which the game supports its intended learning objectives. The final step will involve publishing the game and its results.

5 Conclusion and Future Work

With the growing demand for high-quality software, students need skills required by the software market. However, given the practical nature of SE, approaches centered on theoretical lectures followed by software projects may reduce engagement and motivation. Educational games can complement traditional teaching by providing engaging environments for applying SE concepts in practice.

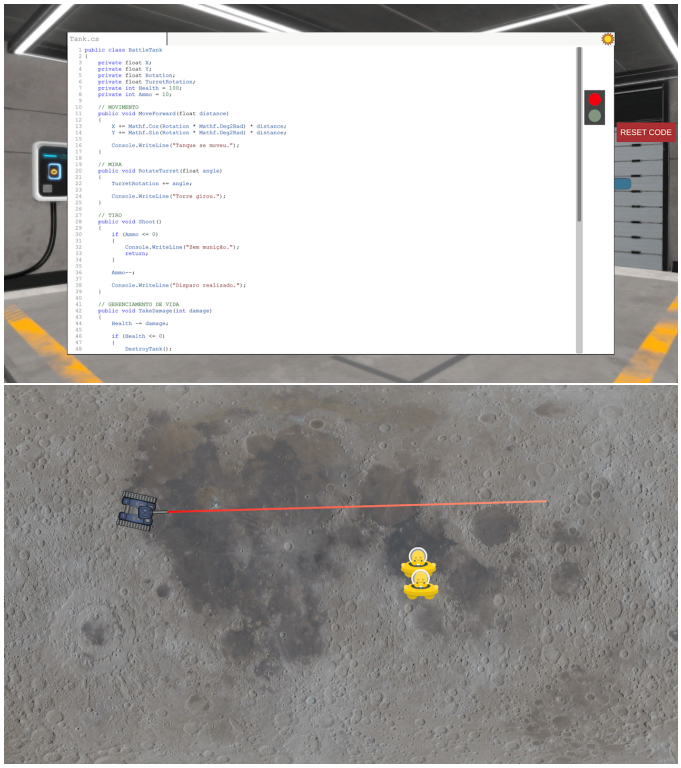

Figure 2: Code editing and alien fight scenes.

This paper presented SOLID Tanks, a SG designed to introduce undergraduates to SOLID principles through code-refactoring activities. By combining practical programming challenges with immediate feedback and rewards, the game provides an engaging environment where students can experiment, reinforce their understanding of software design principles, and receive automated feedback through the proposed assessment mechanisms, potentially reducing the need for manual evaluation by instructors.

For future work, we intend to continue the game's development, conduct playtesting and formal evaluation according to the remaining MEDIEVAL steps, and deliver a fully functional version implementing the proposed features and game design elements.

Artifact Availability

The game's code, Endo-GDC and SGDD are available in the first author's GitHub: https://github.com/miguelsantosan/SOLID-Tanks

Acknowledgements

This work was supported by the National Council for Scientific and Technological Development (CNPq, Brazil), under grant number 131528/2025-4, and by the Carlos Chagas Filho Foundation for Research Support of the State of Rio de Janeiro (FAPERJ, Brazil), under grant number E-26/202.119/2026 - SEI-260003/008755/2026. ChatGPT was used to assist with the development of the Unity game, backend service, and manuscript revision. All outputs were reviewed and validated by the authors.

References

[1] Alan RL Loyola Alvarez and Segundo E Cieza-Mostacero. 2024. Serious Video Game to Improve the Learning of Software Design Patterns. *TEM Journal* 13, 3 (2024), 2557.

[2] Atari, Inc. 1979. Asteroids. Arcade video game designed by Ed Logg and Lyle Rains.

[3] Caner Balim, Naim Karasekreter, and Özkan Aslan. 2026. Automatic multi-language analysis of SOLID compliance via machine learning algorithms. *Information and Software Technology* 192 (2026), 108013. doi:10.1016/j.infsof.2026.108013

[4] Esmail Bonakdarian and Laurie White. 2004. Robocode throughout the curriculum. *J. Comput. Sci. Coll.* 19, 3 (Jan. 2004), 311–313.

[5] Nabila Bousbia, Belkacem Mostefai, Tarek Boutefara, Reda Morsli, and Khaoula Mouheb. 2026. Design of Refactal: A Serious Game for Introducing Advanced Object-Oriented Programming Concepts. *Journal of Educational Computing Research* 0, 0 (2026), 07356331261434779. arXiv:https://doi.org/10.1177/07356331261434779 doi:10.1177/07356331261434779

[6] Diego Castro, Filipe Arantes, and Cláudia Werner. 2019. A tertiary mapping on the use of games for teaching software engineering. *SBC–Proceedings of SBGames* (2019), 1120–1123.

[7] Filipe Furtado, Pedro Henrique Valle, Marcelo Renhe, and Alessandreia Oliveira. 2024. Jogos sérios aplicados ao ensino-aprendizagem de Engenharia de Software: Um mapeamento sistemático. In *Anais do XXXV Simpósio Brasileiro de Informática na Educação* (Rio de Janeiro/RJ). SBC, Porto Alegre, RS, Brasil, 2531–2547. doi:10.5753/sbie.2024.242679

[8] Ivan García, Carla Pacheco, Andrés León, and Jose A. Calvo-Manzano. 2020. A serious game for teaching the fundamentals of ISO/IEC/IEEE 29148 systems and software engineering – Lifecycle processes – Requirements engineering at undergraduate level. *Computer Standards & Interfaces* 67 (2020), 103377. doi:10.1016/j.csi.2019.103377

[9] Mehmet Kosa, Murat Yilmaz, Rory O'Connor, and Paul Clarke. 2016. Software Engineering Education and Games: A Systematic Literature Review. *Journal of Universal Computer Science* 22, 12 (2016), 1558–1574.

[10] David R. Krathwohl. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice* 41, 4 (2002), 212–218. arXiv:https://doi.org/10.1207/s15430421tip4104$_2$ doi:10.1207/s15430421tip4104_2

[11] Marco Kuhrmann, Joyce Nakatumba-Nabende, Rolf-Helge Pfeiffer, Paolo Tell, Jil Klünder, Tayana Conte, Stephen G. MacDonell, and Regina Hebig. 2019. Walking Through the Method Zoo: Does Higher Education Really Meet Software Industry Demands?. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 1–11. doi:10.1109/ICSE-SEET.2019.00009

[12] Robert C Martin. 2000. Design Principles and Design Patterns. *Object Mentor* 1, 34 (2000), 597.

[13] Rodrigo L Motta and José Trigueiro Junior. 2013. Short game design document (SGDD). *Proceedings of SBGames* 2013 (2013), 115–121.

[14] Sabato Nocera, Simone Romano, Rita Francese, and Giuseppe Scanniello. 2025. Software engineering education: Results from a training intervention based on SonarCloud when developing web apps. *Journal of Systems and Software* 222 (2025), 112308. doi:10.1016/j.jss.2024.112308

[15] Marcus Parreiras, Geraldo Xexéo, and Pedro Marques. 2022. Proposta e Estudo de Caso de um Método para Design de Vídeo Games Educacionais. In *Anais Estendidos do XXI Simpósio Brasileiro de Jogos e Entretenimento Digital* (Natal/RN). SBC, Porto Alegre, RS, Brasil, 188–197. doi:10.5753/sbgames_estendido.2022.226084

[16] Giani Petri, Christiane von Wangenheim, and Adriano Borgatto. 2019. MEEGA+: Um Modelo para a Avaliação de Jogos Educacionais para o ensino de Computação. *Revista Brasileira de Informática na Educação* 27, 03 (2019), 52–81. doi:10.5753/rbie.2019.27.03.52

[17] Thiago R. da Silva, Taina J. Medeiros, and Eduardo H. da Silva Aranha. 2015. The Use of Games on the Teaching of Programming: A Systematic Review. In *Proceedings of the Workshop on Experimental Software Engineering (ESELAW'15)*. 474–487.

[18] Bernardo Taucei. 2019. *ENDO-GDC: Desenvolvimento de um Game Design Canvas para Concepção de Jogos Educativos Endógenos.* Master's thesis. Universidade Federal do Rio de Janeiro.

[19] Adilson Vahldick and José Vargas Nolli. 2026. Introducing the Visitor Design Pattern through a Digital Serious RPG Game. *International Journal of Serious Games* 13, 1 (Jan. 2026), 137–157. doi:10.17083/pzmm0y41

[20] Maja Videnovik, Tone Vold, Linda Kiønig, Ana Madevska Bogdanova, and Vladimir Trajkovik. 2023. Game-based learning in computer science education: a scoping literature review. *International Journal of STEM Education* 10, 1 (06 Sep 2023), 54. doi:10.1186/s40594-023-00447-2