A pedagogical approach to teaching game programming: Using the PRIMM approach

Law, Robert

Published in:
Proceedings of the 14th European Conference on Games Based Learning

Publication date:
2020

Document Version
Peer reviewed version

Link to publication in ResearchOnline

Citation for published version (Harvard):
Law, R 2020, A pedagogical approach to teaching game programming: Using the PRIMM approach. in P Fotaris (ed.), Proceedings of the 14th European Conference on Games Based Learning, European Conference on Games Based Learning, Reading, pp. 816-819.
Abstract: Students repeatedly find programming a complex subject to grasp and this can be compounded when teaching within a computer game context. Comprehending the concept of programming beyond the syntax of the chosen programming language can present many barriers for the student to overcome. Students frequently associate the term "programming" with the creation of "text-based" language-specific code neglecting the associated problem-solving activities of developing pseudocode algorithms, computational thinking, problem identification and pattern recognition. Specialising the programming for games can persuade students to neglect problem-solving activities and perceive the programming fundamentals taught in introductory first-level courses as not applicable. Suggesting that the teaching methodology employed at an introductory level does not meet the needs of the student. Various academic studies would suggest there is a need for an approach to programming that brings together as many, if not all, the cognitive aspects of programming - pseudocode algorithms, computational thinking, problem identification, pattern recognition as well as the development of coded solutions. The Predict, Run, Investigate, Modify and Make (PRIMM) approach carefully developed for effective use in schools lends itself nicely for use in a game programming setting. The PRIMM approach is predicated on the spheres of programming research - Use-Modify-Create, Levels of Abstraction, and reading and tracing code for understanding. It provides a framework for the educator to administer "scaffolded" and directed exercises to the student to aid their understanding. Although developed as a general pedagogy for teaching the fundamental aspects of programming, the methodology provides an approach that could improve the teaching and delivery of game programming modules. This paper will discuss an initial attempt to improve the teaching and student understanding of games programming through the application of the PRIMM methodology to the module delivery.

Keywords: Programming, Pedagogy, Methodology, PRIMM

1. Introduction

There is a considered belief that students, regardless of age, find computer programming a problematic concept to comprehend (Teague 2011) and mundane (Mozelius et al., 2013). Gaining “an early understanding” of programming fundamentals is crucial to keep the student engaged and enthused to prolong their programming journey (Shuhidan et al. 2009). These courses can have substantial attrition rates posing a major stumbling block for the students’ academic progression. Shuhidan et al. (2009) suggest that the crux of learning to program is getting the students to understand the "foundation level programming concepts sufficiently early". This should help to alleviate any undue frustration incurred when learning to program. In summary, Shuhidan et al. (2009) believe that establishing a concrete foundation for the novice programmer will allow them a springboard from which they can continue their programming studies. Mozelius et al. (2013) suggest programming modules concentrate on “syntax and algorithms” with little or no thought to the student’s learning style.

The traditional approach to teaching programming has centred around syntax and one specific programming language; rooted in the underlying development of Computer Science over time (Fincher 1999). Fincher (1999) notes the change from teaching programming as a means of enabling the computer "to do something" to a "transferable skill in its own right." Fincher (1999) observes this change has led to the development of alternative "conceptual models and methodologies" employed to the teaching of programming.

Renton (2016) states that teaching students to develop and code games, represents an excellent approach to engross them in programming; however, pedagogical approaches for teaching programming require development. This short paper will reflect on one such recently developed pedagogy known as the PRIMM (Predict, Run, Investigate, Modify and Make) model (Sentance & Waite 2017). This model will be used to encourage students to investigate, design, implement and test their own and others code as a means of developing their programming skills (Mozelius et al., 2013).
2. Literature Review

Muller & Kidd (2014) and Law (2016) assert the essence of programming to be immensely demanding teaching a novice. The subject of programming can be thought of as an art form expecting the student to engage with it regularly to completely absorb and become attuned to its nuances (Murphy et al. 2016)(Fincher 1999). Teague (2011) also asserts from her review of literature that “lots of hands-on practice and experimentation is especially important for novice programmers”. As students begin to construct their knowledge of programming from “their own practical experiences” and not from an acceptance of the material presented in books and lectures (Teague 2011).

Winslow (1996) identifies three strands: first, the starting point for moving from novice to expert lies with the application of “general problem solving” underpinned with “basic knowledge ... of the field”; second, expertise develops and grows with “practice”; finally, the repetition of the aforementioned two steps with the continual adding of new facts and problems. Surveys such as that conducted by Robins et al. (2003) suggest that teaching should not only concentrate on introducing new features of a chosen programming language, it should include the integration and use of the programming language features emphasising the fundamental topic of elementary program design. Spohrer & Soloway (1986) assert the notion that novice programmers have issues with programming due to their inability to “plan composition problems” (construct a program from its constituent parts). Therefore, problem-solving is an essential skill for students to learn. Xinogalos (2016) corroborates this research suggesting that problem-solving consists of four parts: “Understanding the definition of a problem”; “Selecting the appropriate structures (if, if/else, switch, for, while, ) for solving a problem”; “Developing an algorithm for solving a problem” and ”Transferring the algorithm to the programming language”. In his analysis of findings Xinogalos (2016) identifies “modularization”, generation of algorithms, and implementing algorithms in a programming language as the major causes of adversity for novice programmers. In their evaluation of the results from their study of novice programmers, Lahtinen et al. (2005) identify three key aspects that the surveyed students regarded as hard concepts to understand: “understanding how to design a program to solve a certain task”, “dividing functionality into procedures” and “finding bugs from their own programs”. Thus, they hypothesize, students have to understand more about the overall workings of the program than just the aspect they are working on. Designing a teaching strategy that will be “challenging and interesting” may prove prohibitive as the class composition will comprise of students of varying levels of programming knowledge (Lahtinen et al. 2005).

Murphy et al. (2016) posits the role that the school curriculum can play when establishing a sound programming foundation for the student. Establishing a computing curriculum in UK schools has introduced the teaching of programming to children at an early age; initially, through block-based languages with the need to switch to text-based languages as their learning progresses (Murphy et al. 2016). Scotland’s Curriculum for Excellence (CFE) puts an emphasis on the utilisation of computers and various learning styles and approaches in a classroom setting leading Wilson et al. (2013) to undertake a study of primary school children learning programming via game development using Scratch. Wilson et al. (2013) concluded from their 8-week study that the children achieved “some programming constructs”.

Sentance & Waite (2017) have developed a teaching approach to programming for school teachers called PRIMM - Predict-Run-Investigate-Modify-Make (see Figure 1) which could help to mitigate the issues arising from the introduction of text-based languages. PRIMM builds upon three particular spheres of research; “Use-Modify-Create, Levels of Abstraction and reading and tracing code for understanding” (Sentance & Waite 2017). Moreover, the block model proposed by Schulte (2008) has helped to influence aspects of the PRIMM model (Sentance et al. 2019). The PRIMM model offers the student both “challenge and entertainment” which Mozelius et al. (2013) believe are important factors for providing a rewarding learning platform.

![Figure 1 PRIMM - Predict, Run, Investigate, Modify, Make](Sentance & Waite 2017)
The "use-modify-create" framework developed by Lee et al. (2011) maps to the three stages of a "students' cognitive and practical activity in computational thinking." Underpinning the concept is the assumption that "scaffolding increasingly deep interactions" will advance the attainment and growth of computational thinking (CT) skills (Lee et al. 2011). The "Use" stage allows the student to use code produced by someone else exposing them to programming concepts and code implemented to produce a working program. The "Modify" stage allows the student to change, enhance and adapt the existing code through incremental refinements of greater complexity; illustrating a deeper understanding and comprehension of the original codebase. The "Create" stage allows the student to apply their newly developed skillset to projects of their design bringing together the three integral components of CT – abstraction, analysis and automation (Lee et al. 2011). The student's journey through this cycle should provide the student with a level of inquiry that encourages and maintains development; keeping apprehension at a minimum. Advice for educators using this three-stage is not to take it "too literally" as there is no precise interval at which each stage can transition; students may well oscillate between stages (Lee et al. 2011).

Teague & Lister (2014) found first-year students ability to progress with programming tasks escalated as their capacity to think abstractly developed. Venables et al. (2009) note the ability to trace, explain and write code supplement each other flourishing in parallel. Concluding the ability to write code can open up possibilities for enhancing code tracing and explanation proficiency. Hence, writing some code should provide the wherewithal to investigate that code, authenticating its validity and subsequently "explain" the code to oneself." (Venables et al. 2009). The Investigate, Modify and Make of PRIMM shares commonality of purpose with Lahtinen et al. (2005) observation that teaching and learning resources should encourage the student to "develop program generation, modification and debugging skills." through the use of "exercise sessions", "practical sessions" and "example programs" thus underscoring the pedagogy of learning by doing.

3. Applying the PRIMM Model

Adopting the model for the second year Games Programming 1 (GP1) module which comprises of Game Software Development (GSD) and Game Design (GD) students. This introductory module introduces C/C++ and SDL for the development of simple 2D games. Students had completed an introductory games module using Unity with C# in their first year. The GSD students also completed an introductory programming module using Java. The GSD students have a basic underpinning in programming principles but the GD students may not necessarily be at the same level of proficiency. The PRIMM model offers a solid framework to introduce programming to novice programmers in a structured and well-defined manner.

The strategy for delivering GP1 is to refresh the students understanding of the basic building blocks of programming (sequence, selection and iteration) before learning basic game programming techniques and subsequently 2D graphic techniques. The module reinforces the basic programming building blocks before illustrating how a combination of these building blocks can produce basic games using C/C++ in console mode. This illustrates the transition from simple console games to simple 2D graphical games using the code developed for the console as the core of a 2D graphical game. The module contains three sections: reiteration of the basic programming building blocks, console games and consolidation, and introduction to 2D graphics and game programming. Each section implemented using the PRIMM model.

4. Conclusion

Having implemented the PRIMM model for a complete year 2 module several observations have been made. Firstly, the time needed to produce material fitting the model is considerable. Secondly, the level of detail applied to the materials produced for the students has a fine grain of granularity. Thirdly, student engagement is required. Students engaging with the material appeared, albeit anecdotally, to have a better understanding of the programs presented. Also, appearing less anxious about coding, in general, producing a good level of programming coursework.

The PRIMM model offers a structured framework rooted in three well-defined areas of research: Use-Modify-Create, Levels of Abstraction, and reading and tracing code for understanding (Sentance & Waite 2017), as such, it is built on a sound pedagogical platform. This sound pedagogical platform is evident in its well-defined structure and ease of implementation. Quality materials will need to be developed, tweaked and rewritten.
regularly. This is as much to do with varying cohorts as it is to keep material current with improvements in the programming language being used.

References


