Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education

Teresa Terroso
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Mário Pinto
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Abstract

Learning how to program can be a cumbersome task even for students who enroll in courses in the Computer Science field. It is well documented that computer programming courses have high failure rates and high drop out. Even at the initial stage of computer introduction courses, novice students often reveal difficulties and strong reactions to this subject. However, computer programming has been recognized as an essential skill and a necessary element in education in many different areas. This work reflects on the experience provided by teaching a Creative Programming course, being held as part of a Master's degree curriculum in School of Media Arts and Design (ESMAD), at Polytechnic of Porto (P.PORTO), in Portugal. The students' background is not uniform, therefore pedagogical learning strategies had to be adapted to these multidisciplinary backgrounds to foster student attention and interest, as well as being able to achieve the goals of teaching the fundamentals of computer programming. This article reflects on the strategies to teach programming for non-informatics: drifting from the traditional functional way, like developing a program or product to solve a problem, to the usage of creative coding and generate interactive animations, while simultaneously achieving the ambitious goals of learning programming concepts and paradigms.

2012 ACM Subject Classification Computing methodologies; Applied computing → Computer-managed instruction

Keywords and phrases creative coding, programming learning, teaching methodologies, higher education, visual applications, interactive graphics, open-source tools

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.13

1 Introduction

It is well known that many students have difficulties in learning computer programming since it is a complex activity, requires a lot of practice, and traditional teaching approaches have not been able to respond effectively [12, 4, 1, 10].

Different reasons are enumerated and can be arranged into five major groups: teaching methodologies (non-personalized methods, static materials for dynamic programming concepts, more focus on teaching a programming language rather than promoting problem-solving), study methods (rather than practice intensively, students focus on reading books or watching tutorials and memorize formulas or procedures), student's abilities (generic lack of problem understanding, relating knowledge and infer a solution), the nature of programming (abstract concepts and complex syntactic details of many programming languages) and psychological effects (usually taught at the beginning of a higher education course and negative connotation associated with programming).

The high dropout and failure rates in introductory programming courses are a universal problem that motivated many researchers to propose methodologies and tools to help students: game design and development, programmable physical/tangible tools, project-based learning, gamification, automatic code evaluation systems, and so on [7, 10].

Although these approaches may motivate students to learn to program, most are oriented to computer science programs and thus special attention must be taken to students with different academic profiles and/or professional backgrounds. Creative coding as been already discussed in some works as an additional approach concerning computer science education, therefore this work contributes to the topic by providing further data supporting that it can be an additional path in teaching the basics of computer programming.

2 Creative coding

Creative and problem-solving competencies are part of the so-called 21st-century skills and are considered crucial to succeed nowadays [14]. Everyone, not just students who major in computer science, can benefit from thinking like a computer scientist.

As stated in [3], creativity is "the tendency to generate or recognize ideas, alternatives, or possibilities that may be useful in solving problems, communicating with others, and entertaining ourselves and others". By extension, creative coding can be understood as a type of programming where students can use the computer to produce visual animations to self-express themselves rather than focus on functionality. As learning computer programming often emphasizes technical detail over creative potential, creative coding supports increasing fluency with computational thinking, build upon creativity, imagination, and students' personal interests. According to [8], computer technology "is not a tool; it is a new material for expression". This intersection between arts and technology has received attention in both professional and educational fields [9, 6]. New art forms are being developed through the medium of code daily, and the creative coding community is growing rapidly.

Creative coding, as a programming practice geared towards artistic content production, has already been found effective in learning computer programming [5, 15, 2]. This work can be placed together within diverse efforts for introducing programming in an artistic manner to students, beyond the engineering perspective.

This work reflects on the findings from teaching a one semester-long Creative Programming course, taught on the first year of a Master degree in Interactive Media and Systems at ESMAD, P.PORTO. The main goal of this course is to teach the elementary concepts of computer programming applied to the different contexts of multimedia digital arts, by presenting a creative programming language as a tool for artistic expression. At the end students should be able to design and produce programs for the generation of audiovisual content for performances, multimedia installations, or web applications. Most of its students do not have an inherent affinity towards programming. As a course in a master degree, the classes are composed by students with very different educational or professional backgrounds: from artists to designers, from physiotherapists to multimedia students. Traditional ways of teaching programming (lectures combined with exercises, assessed by only one project and a final exam) had been tried, but students found that it was hard for them to build the link between theory and practice. Students are often eager to put the just learned knowledge into practice, if not immediately, at least as quickly as possible. Most of them have passion in visual designs so, they would like to see immediately their creations "move". The followed approach focused less on manipulating and interacting with numerical or text data but more on digital animations (from building graphics, to audio or video manipulation), that have been found to be at their interests as young adults and also making them more amenable to a heterogeneous environment audience.

To attend the course no programming experience is required. However, students should be comfortable using a computer, to perform simple tasks like installing applications and working with files. Within the efforts of creating and expressing themselves through digital media are the building blocks for introducing the essentials of programming. Over the semester, with a total of 30 contact hours, students develop a portfolio of digital animations using P5.js (an interpretation of Processing written in JavaScript) and employing basic computing structures typically taught in traditional Computer Science courses.

3 p5.js

Different creative tools (like Processing or p5.js) have been integrated into the curriculum of several education institutes, since they foster experimentation and enable the use of computation to express ideas with visually engaging sketches or interactive installations. An extensive list of curated tools and other resources for creative coding can be found in [13] and table 1 summarizes some of the most found in literature regarding creative coding.

Table 1 Tools for Creative Coding.

Frameworks for desktop development		
	Language	Mainly used for
Processing	Java	General purpose (IDE for visual arts)
OpenFrameworks	C++	General purpose (no IDE, toolkit for creative coding)
Cinder	C++	Professional-quality creative coding (uses OpenGL)
Libraries for web development		
	Language	Mainly used for
p5.js	JavaScript	General purpose (alternative for Processing)
three.js	JavaScript	3D graphics (simplifies working with WebGL)
Paper.js	JavaScript	Vector graphics scripting

In the course presented in this work, the visual programming tool is p5.js, a JavaScript library, as a web-based alternative to Processing, but with the same goal of making coding accessible for artists, designers, and overall beginners.

Processing was created in 2001 as a language for teaching art students how to program and to give an easier way to work with graphics to a more technical audience. P5.js emerged as a more accessible version, but still adhering to the syntax and conventions of Processing. Open-source and with a strong community of contributors, p5.js has capitalized on the web browsers' features and ubiquity. Compared to plain JavaScript, p5.js provides a more readable and clean code, making it easier to work with the Canvas HTML element and implement frame-by-frame animations, Table 2.

Table 2 Code snippets to illustrate some differences between plain JavaScript and p5.js.

Drawing a circle		**Frame animation**	
Plain JavaScript	p5.js	Plain JavaScript	p5.js

c.beginPath();
c.arc(x,y,r,0,2*Math.PI);
c.closePath();
	circle(x,y,r);	function draw() {
		/*some code*/
		requestAnimationFrame(draw);
		}
		draw();
			function draw(){
			/*some code*/
			}

For students who are learning how to program, p5.js provides encouragement and motivation with the immediate visual feedback. P5.js adds custom features related to graphics and interaction and provides easier access to native HTML5 features already supported by the browser. Several libraries extend p5.js even further, like DOM manipulation, webcam capture, image/sound processing and so on. Experimenting is made easier as there is an online editor[1] available on their website, allowing for a quick start to the library.

Choosing between Processing and p5.js was not difficult, even with the shortlist of bibliography or online examples. Having to choose between Java (Processing) and JavaScript (p5.js) as the base programming language, the choice fell towards the latter as its dynamic language features, and ability to merge functional and object-oriented programming techniques make it particularly well suited to exploratory creative practice. It has also been identified as an excellent language for introducing computer science students to programming [11].

P5.js follows the concept of sketching as its programming method, where it basically consists of quickly throwing 2D and 3D graphics for rapid prototyping of a dynamic visual work, similar to how drawn sketches are created by artists. This contrasts with the standard and disciplinary approach of software engineering for software development where first specifications are created, then implemented and evaluated. Despite its easiness to start to work with, it is a full-featured library capable of rendering stunning graphics and animations, using a simplified syntax and graphics programming model.

4 Course design

The course presented in this work consists of the following topics, for the total duration of one semester:

Course Introduction Introduction to Computing: structure, logic, syntax and algorithms. What is creative computing? Historical context, artistic references and examples.

Drawing primitives 2D and 3D coordinate systems. Simple shapes: points, lines, shapes, curves, text, images and 3D objects. Drawing and styling.

Control structures Variables and data types, operators, expressions, conditionals and loops.

User interaction Mouse and keyboard events, basic animation.

Functions Procedural abstraction, declaration and invocation of functions, parameters and return values.

Arrays and Objects Introduction to arrays, lists and indexing. Object-Oriented Programming (OOP): objects and classes.

Mathematics and Physics Basic trigonometry, direction, oscillation, acceleration, inertia and friction.

Creative coding examples Transformations, iteration and randomization.

Each topic was structured around 1-2 weeks of lectures and exercises, complemented with creative coding extra-class assignments. Although the textbook contents and topics ordering follows more or less the same as traditional introductory computer programming courses, the key concepts are most of the times driven by the assignments themselves or appear as tools to respond to the students' wanderings 'what if we would like to...?'.

During classes, the explanation of programming concepts is supported using visual graphics, Figure 1. In experience gained from teaching this course, it was found that using program visualization techniques helps grab the students' attention and support learning key programming concepts. Examples taught in classes serve as entry point to analyze the logic, decompose the algorithm within, or spotting for patterns or similarities. Exercises usually

[1] `https://editor.p5js.org/`

have an inspiration as starting point, were students are challenged to experiment and create something new, or debug to find and fix errors or even to collaborate by working together to find a solution. Some researches have demonstrated the impact of visual programming tools in upgrading pass rates and overall improvement when compared with other teaching interventions to facilitate student learning like taking an intensive preliminary training course or workshop, [7]. Like many, and from early on, the professors of this course were faced with problems aroused from the fact that students enter the Master degree with widely different experience levels with key course topics. If the material is covered too slowly, those with greater experience get bored and lose interest. If the material is covered too quickly, those with less experience get lost and feel incompetent. That led them to promote participation in short introductory programming courses, available for free in various e-learning platforms, targeting students with little or no background in programming. This intervention is still in practice today but can be improved since it is not to mandatory and the professors cannot control the course contents, acting only as curators in the selection of courses.

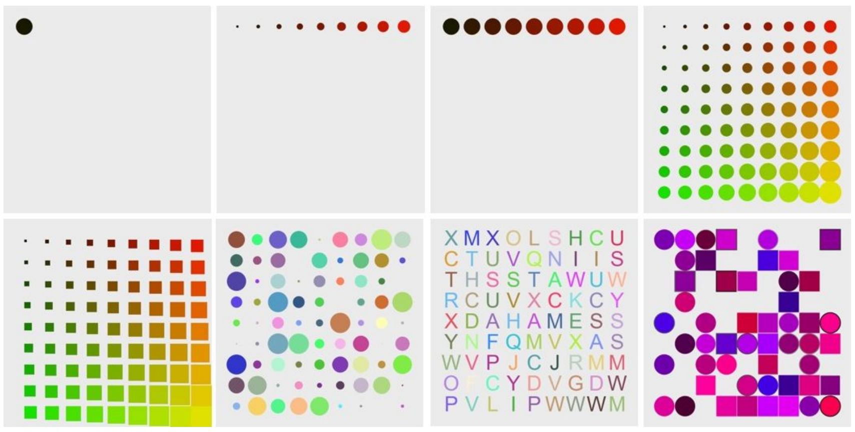

Figure 1 Examples presented during classes to visually demonstrate key programming concepts like loops, conditionals and operators. Top to bottom, left to right, from the simplest static sketch (no loop iterations and no conditionals) to showcasing different variations, by gradually introducing loops, nested-loops, conditionals, user inputs and animations.

Based on this work experience, the usage of media computation (mostly computer graphics, but also including image or sound processing), the foundation of creative programming, gave the students a context in which they already find the computers useful, which combined with open-ended exercises and assessments provided creative activities and restrained the stereotypes of computing as boring and anti-social, Figure 2.

Over the course of a semester, students are encouraged to develop further and share their sketches (in our school's learning management system or on public websites like openprocessing.org), so they can learn from each other's work.

4.1 Assessment

Students are asked to develop two projects during the course. The first project, developed individually, focus on simple and short 2D animations, using functional programming, where the theme is set by the professor, Figure 3. The second is a collaborative project with another course of the master's degree, Interfaces and Interaction Design, and it is developed in small groups (2 to 3 students). In this last project, students develop an interactive installation or application on a topic chosen by them, with a mandatory requirement of using OOP. In

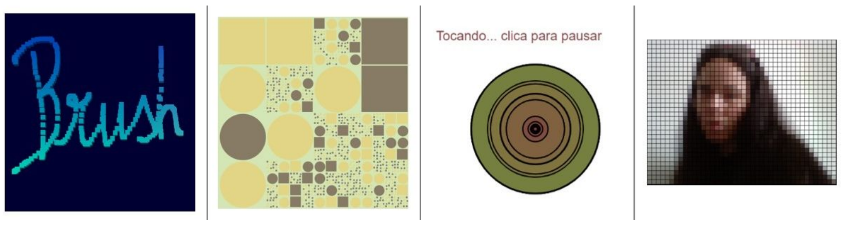

Figure 2 Examples of computer art techniques: from left to right, virtual brushes, recursive structures, sound visualization, pixel art.

this sense, students must develop an intuitive interface (with the input of user information through a camera or computer user interfaces, like mouse or keyboard) and/or tangible (through the Arduino micro-controller) for the public to be able to generate and manipulate audiovisual content. The format of the projects may vary from web applications, to 2D or 3D games or artistic installations, Figure 4.

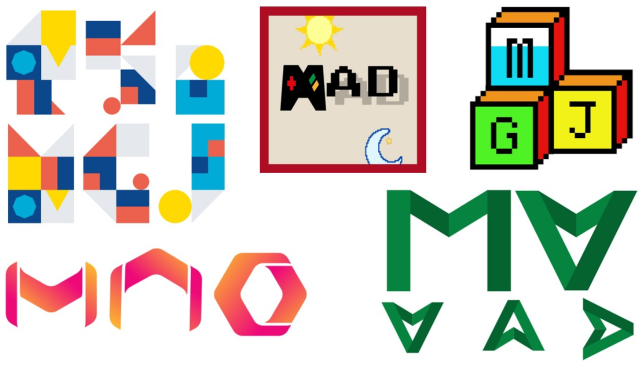

Figure 3 Project 1 example: creation of an animated logotype for one of our school's annual event, the Mad Game Jam.

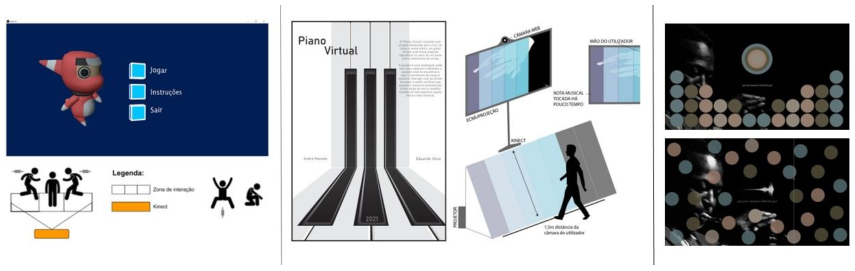

Figure 4 Work examples for Project 2: from left to right, 3D game (player controlled by body movement), virtual piano (audio and visuals controlled by hand movement), web music player (visuals controlled by sound samples).

Assessment of the students includes traditional instruments such as critiques and presentations. The critique process comes out of the classroom, where student work is assessed primarily through discussions including both technical aspects of the projects, such as a review of the source code, and also basic aesthetic issues. During final presentations, students are asked to perform auto- and hetero-evaluations.

5 Preliminary results

In the past three editions of this course, from a total of 39 enrolled students, only 6 dropped-out. From the remaining 33 students, only 4 (little more than 12%) had some previous knowledge about computer programming. All students were asked to answer a small survey about their findings, but only the students that completed the course answered, gathering a total of 16 responses (about 48%). No extensive statistical data analysis is presented in time for this work, since the surveys have suffered some alterations from year to year and the sample size is relatively small. However, some observations can be pointed out from the students' answers:

Course rating Students' were asked to grade the course on a 1 to 5 scale, with 1 as "it did not meet my expectations at all", 3 equals "it met my expectations" and 5 as "it exceeded all my expectations". The average rating was 3.75, having increased when the course assessment has shifted from individual tests (with theoretical questions and small practical exercises) and one group project, to two projects (one individual and the other developed in small groups).

Main strengths Some of the students' answers included "Learn to make generative art, and give yourself freedom with code", "Learning creative programming allowed me to start exploring areas that I had no knowledge of until now" and "I found myself entertained with some of the homework exercises because it was fun". 78% of students responded very favorably to the context of art when learning how to program and 43% of those claimed to spend extra time on assignments because they enjoyed it. Not so significantly, but a few students stated that the course instilled in them the desire to further explore the area of programming.

Main weaknesses A significant percentage of students (around 87%) pointed out the reduced in-class time to consolidate concepts and a few would have liked to explore some topics more in-depth. 24% of the respondents, even with the online course taken at the beginning, still struggled with the programming basics and "ended up confusing everything".

6 Conclusion and Future Work

The work presented in this paper reflects the usage of creative coding in Higher Education, as a way to overcome the difficulties of learning how to program, especially in classes that have different learning backgrounds, therefore fostering their interest and demystifying some of the negative connotations associated with programming. Creative coding can act as an approach to minimize the issues of code illiteracy and provide the tools for empowerment in this digital era. As a way to create art through code, creative coding can also be understood as a path for learning programming through creative projects. Therefore, it can be encouraged as a teaching tool, as a new way for exploration, computational reasoning development, and critical reflection about programming. The authors will continue to explore the opportunity to refine the course design and develop course materials (e.g. using the power of visual environments, that rather than just showing code structure and results, can offer dynamic data behavior) and accumulate student feedback, so that this work findings can be better substantiated and compared with other teaching interventions to facilitate student's learning in introductory programming courses.

References

1 Yorah Bosse and Marco Aurelio Gerosa. Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage. *ACM SIGSOFT Software Engineering Notes*, 41:1–6, January 2017. doi:10.1145/3011286.3011301.

2 Vincenzo Fragapane and Bernhard Standl. Work in progress: Creative coding and computer science education – from approach to concept. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 1233–1236, 2021. doi:10.1109/EDUCON46332.2021.9453951.

3 R.E. Franken. *Human Motivation*. Brooks/Cole Publishing Company, 1994. URL: https://books.google.pt/books?id=hfW6AAAACAAJ.

4 Anabela Gomes, Cristiana Areias, Joana Henriques, and Antonio Mendes. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, 42:161–179, July 2008. doi:10.14195/1647-8614_42-2_9.

5 Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pages 247–252, 2012. doi:10.1145/2157136.2157214.

6 Edmund Harcourt. Exploring the intersection between art and technology, May 2021. URL: https://hackernoon.com/exploring-the-intersection-between-art-and-technology-en1s34fd.

7 Arto Hellas, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, July 2014. doi:10.1145/2632320.2632349.

8 J. Maeda, R. Burns, Thames, Hudson, and Massachusetts Institute of Technology. Media Laboratory. *Creative Code*. Thames & Hudson, 2004. URL: https://books.google.pt/books?id=VeO6GwAACAAJ.

9 Kylie Peppler and Yasmin Kafai. Creative coding: Programming for personal expression. In *8th International Conference on Computer Supported Collaborative Learning (CSCL)*, pages 76–78, June 2009.

10 Ricardo Queirós, Mário Pinto, and Teresa Terroso. Computer Programming Education in Portuguese Universities. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 21:1–21:11, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.21.

11 John Resig. Javascript as a first language, December 2011. URL: https://johnresig.com/blog/javascript-as-a-first-language/.

12 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.

13 Terkelg. Awesome creative coding. URL: https://github.com/terkelg/awesome-creative-coding.

14 Joke Voogt and Natalie Pareja Roblin. A comparative analysis of international frameworks for 21st century competences: Implications for national curriculum policies. *Journal of Curriculum Studies*, 44(3):299–321, 2012. doi:10.1080/00220272.2012.668938.

15 Zoe J. Wood, Paul Muhl, and Katelyn Hicks. Computational art: Introducing high school students to computing via art. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 261–266, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2839509.2844614.