Scaffolding CT via Point-And-Click and P5.js

Andrea Valente
Department for Media, Design, Learning and Cognition,
University of Southern Denmark,
Kolding, Denmark
aval@sdu.dk

Emanuela Marchetti
Department for Media, Design, Learning and Cognition,
University of Southern Denmark,
Kolding, Denmark
emanuela@sdu.dk

Edward Abel
Department for Media, Design, Learning and Cognition,
University of Southern Denmark,
Kolding, Denmark
abel@sdu.dk

***Abstract*— *This paper investigates how to provide meaningful scaffolding to bachelor humanities students, to enabled them to acquire Computational Thinking (CT) technical skills, and in particular basic programming competences. Two of the authors have been involved in the re-design, implementation and execution of a basic programming and CT course, offered to first-semester students as part of the Information science, IT and interaction design bachelor program, at the University of Southern Denmark (SDU). The central problem we faced in restructuring our introductory course, was finding a game genre that could support creative coding for beginners, be motivational and recognizable by the students, and would work with our use-modify-create learning approach. Our findings suggest that point-and-click games are an effective way to provide scaffolding and ease non-technical students into P5 programming. The genre has good expressive power, and the students were motivated because they recognized and could relate to the games they worked on. Future work will address students' problems with scaling up the point-and-click games.*

Keywords—Computational Thinking, Scaffolding, Humanities, Introduction to Programming, P5.

I. Introduction

In recent years, introductory computer programming courses at universities are increasingly becoming part of programs outside of engineering facility, such as within the humanities and social science faculty, and are being refashioned to include explicit consideration of Computational Thinking (CT). Two of the authors has been involved in the implementation and execution of such a course, entitled *Basic programming and CT*[1], offered to a dozen first-semester students in the fall of 2022, as part of the *Information science, IT and interaction design* bachelor program by the University of Southern Denmark (SDU). The goal of the course, recently extended in size by 33% to 13 3-hour lectures, is to introduce the first semester bachelor students to programming. The first half of the course uses MIT's Scratch (https://scratch.mit.edu/) to explore basic CT concepts like problem decomposition, and algorithms designs, and their programming counterparts in Scratch: sequence, choice, iteration, and function declaration and calling, as well as Scratch-specific animation commands, controls, and the online IDE. Software development practices are also introduced, like testing and debugging, and principles related to iterative software development. These concepts and practices are in line with [1] and [2]'s definition of the central aspects of CT. Scratch was chosen to support learners, helping them start coding; as stated in [3]: *"[...] blocks to text progression has been shown to be beneficial to just jumping straight into text based programming"*. The second half of the course introduces JavaScript via the P5.js library [4]. P5 was chosen to leverage on creative coding, and allow students to create graphic and interactive programs from the very beginning, with little code, easing them into textual programming. Because of the structure of the first year of the education line, discussions about data types, scoping, and functions should be kept to a minimum. Consequently, we should not introduce JavaScript arrays, and focus instead on primitive types.

The aim of our study is to reflect on how to provide meaningful *scaffolding*, or support, to beginner programmers from non-technical curricula. A central problem we faced in restructuring the course, was finding a good *problem domain* that could support creative coding for beginners, be motivational and recognizable by the students, and would work with our use-modify-create learning approach. And since we want our students to practice creative coding, which involves interaction and graphics, the problem became: finding a *game genre* to scaffold CT. The next section presents related work and ideas on the possible strategies teachers can adopt to organize their materials. Section III introduces the point-and-click game genre, shows how it can be used to support beginners, and discusses our approach to present and implement this genre of games in P5. Section IV discussed our findings and reflections, based on the reaction of the class during the semester and the code the students delivered for their final exam. Conclusion and future work close the paper.

II. Related Work

Our study is grounded within the area of CT, intended as the "thinking" behind the computational problem solving performed with a computer through the creation of algorithms [5], using existing programming languages and tools. More in details, the skills targeted in the course refers mainly to algorithmic thinking and programming and could be grouped within the umbrella categories of abstraction, generalization, decomposition, algorithms (sequencing and control flow) and debugging (as in [5], [6] and [7]). Introducing basic programming skills outside of technical curricula has become a popular research topic as well an internationally spread practice ([5] and [7]). A main challenge for this practice is to introduce

[1] Translated from Danish "Grundlæggende programmering og Computational Thinking".

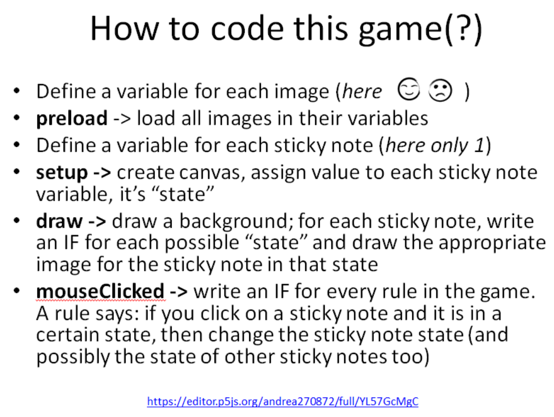
programming as a required skill within an education that has other target subjects: programming is not the main goal for the learners. Hence, their commitment to learning programming might be on the average weaker than technical students. This aspect is investigated by [6], which looked at the challenges of *student teachers* in acquiring CT skills. Interestingly, student teachers see themselves as medium to high proficient in their CT skills. Broader interdisciplinary definitions of CT have been proposed, starting from Wing [1] and [8], including designing systems and understanding human behavior. However, these definitions are often shallow and not especially helpful in educational contexts [7], where the goal is to enable learners to acquire technical knowledge in understanding algorithms and coding, with the societal goal of democratizing the making and use of digital technologies [9]. Therefore, an emerging question regarding learning and teaching CT is how to achieve forms of meaningful scaffolding for technical CT skills for non-technical students. By scaffolding we mean any kind of pedagogical approach, activities, tools, and guidance that teachers can provide to enable learners to acquire new knowledge [10]. The concept of scaffolding has become popular in various educational studies, and it is grounded on a construction metaphor, indicating the use of *"a temporary structure to support and protect the construction of a building"* [10] which will be dismantled at its completion. In this sense, scaffolding in teaching and learning is intended as a system of "temporary guidance" provided by the teachers and negotiated with the learners, changing over time and to be removed once the learning goals are met [10].

The study of supporting learning of CT for non-technical students deal with different forms of scaffolding, for instance [11] investigates a series of seven strategies: load reducing, schema activation, structured based, generative, guided discovery, modeling, and teaching thinking. The first, *Load-reducing*, is aimed at preventing cognitive overload to facilitate the learners to focus on specific topics or procedures while acquiring new knowledge. A typical example is the use of block-coding and remixing code provided by the teachers, to enable beginner programmers to familiarize with coding and to focus on problem-solving, avoiding the challenges of dealing with the syntax of specific programming languages, both strategies were adopted in our studies. This first strategy is in line with the use-modify-create approach to scaffolding, which is typical of the Danish education system. The second scaffolding strategy is called *Schema-activation* and it requires to contextualize new knowledge within the learners' existing understanding. This is in line with Schön [12] and his notion that learning involves the creation of a system of cases, which can be recalled when dealing with new knowledge and new situation to be solved. The third strategy is called *Structured-based* and focuses on enabling learners to "manipulate concrete objects" for better understanding of more abstract concepts or rules. This strategy matches the constructionist playful approach proposed by Papert [13], aimed at facilitating learning through hands-on playful experiences, through sociomaterial engagement and trial-and-error experiments. The fourth strategy is called *Generative* and it requires learners to generate connections between what they know and what they learned. According to Schön [12] this happens through forms of *reflection in action*, while learners are asked to solve specific problems applying what they know and practicing the new knowledge that they are supposed to learn. The fifth one is called *Guided-discovery* and it deals specifically with strategic scaffolding from the teachers' perspective, so to enable the learners to acquire increasing level of freedom in exploring given problems, and identify relations, patterns and underlying rules and principles. The sixth strategy is called *Modeling* and it emphasize demonstrating or explaining the steps involved in solving a given problems. This can include the creation of worked-out examples to be presented to the learners. Understanding the steps involved in algorithmic problem solving is a key competence in CT and learners are required to be reflective about their process [2], this was also a goal for the course discussed in our paper. The seventh and last strategy identified by [11] is called *Teaching Thinking* and deals with explicitly teaching specific metacognition to students, asking them to describe their coding to others and eventually convince another learner of the correctness of their solution through activities of peer learning.

III. From Stick&Click to P5

The need to introduce programming practices and concepts to students outside the technical subjects is spawning new programming environments, languages and libraries that ease beginners into programming, or at least coding ([9] and [11]). In recent years the authors have worked on tools to provide creative scaffolding to for beginners (e.g. in Python [14]) but also in tools that reduce coding to a minimum, while still enabling learners to create interactive digital contents (as in [6]). StickAndClick ([15] and [16]) is one of these tools, and it combines problem-solving with the creation of digital content, in order to foster critical thinking and creativity. Implemented as a special kind of online visual editor, this digital tool is meant to support teachers and pupils by proposing a minimalistic, asset-based redefinition of coding, focusing on the creative and design-like aspects of CT. In StickAndClick learners can design, implement and run simple games, starting from their visual assets, i.e. images. The central mechanic of the games is clicking on an image to change it to another; after the assets are loaded into a new project, visual rules can be defined, expressing before-after semantics by examples. A rule can specify that when clicking on a certain image (e.g. a cat asleep), that image will change into another (a cat awake), but there can also be a condition on other images on the screen. The paper [16] explores the range of games that can be implemented with a tool like StickAndClick, and concludes that even without being Turing complete, StickAndClick's visual rule-based language can describe quite complex games, including point-and-click, turn-based and platform games.

However, even if StickAndClick was not aimed at teaching programming, it already suggested that a simple mechanics like clicking on an image, in conjunction with simple context-aware before-after rules, could result in complex games, possibly restricted to the point-and-click genre. Interestingly, the StickAndClick prototype was implemented using P5, so that it could be easily tested online. It was therefore decided to re-analyze the basic ideas behind StickAndClick as a scaffolding tool, simplify them even further, and create a skeleton code in P5 that could: allow our bachelor students to create simple games with little coding (1 or 2 pages of P5), work without requiring arrays or other kind of advanced data-types, and keep the central ideas from StickAndClick, which are clicking as the main game mechanics and rules based on the images currently on the screen. However, given that our code examples cannot use arrays or iterate over collections of any kind, we could not give the learners a single code skeleton to be used as a game engine, and that would work simply by changing the list of images to load, or redefining some data-structure to express different rules for a particular game. Instead we defined a **recipe**, a meta-level algorithm to generate any StickAndClick-style game in P5. We still presented the students with a few examples of point-and-click games implemented in P5 following the recipe, and following the use-modify-create approach, after we played with the games and looked at the code, we gave them time to make sense of what they would change to customize these exemplar games (see Schön's exemplars in [12]). Only later they were asked to create their own games, using the exemplars, the recipe, and providing their own images. In this sense, this game recipe acts as a scaffolding resource [10], enabling learners to start coding; the recipe itself will then be used as a simple reference as they progress, making their own games.

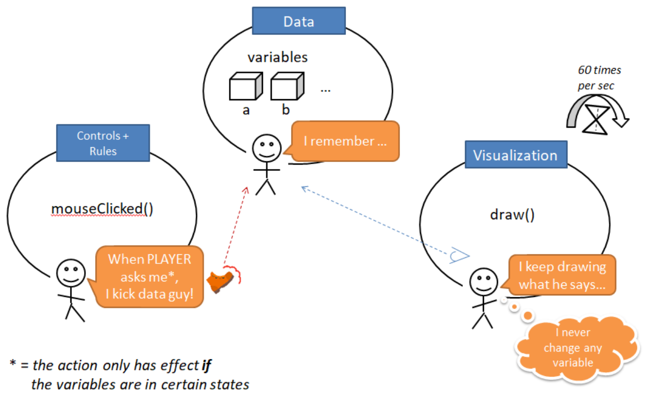

How to code this game(?)

- Define a variable for each image (*here* ☺ ☹)
- **preload** -> load all images in their variables
- Define a variable for each sticky note (*here only 1*)
- **setup ->** create canvas, assign value to each sticky note variable, it's "state"
- **draw ->** draw a background; for each sticky note, write an IF for each possible "state" and draw the appropriate image for the sticky note in that state
- **mouseClicked ->** write an IF for every rule in the game. A rule says: if you click on a sticky note and it is in a certain state, then change the sticky note state (and possibly the state of other sticky notes too)

https://editor.p5js.org/andrea270872/full/YL57GcMgC

Fig. 1. The recipe to code any simple point-and-click game.

The recipe is depicted in Fig. 1, and to make sense of the 3 portions of the code that it describes, we presented the students with the diagram in Fig. 2. Note that since we cannot use collections, the students have to analyze their game and decide how many variables to create, with the suggestion that each image is connected logically to 2 variables: one to contain the actual image, loaded by P5 in the *preload()* method, and a variable named after the image, that remembers the state of that image. In the first exemplar of such games that we introduced to the class, the canvas shows a single image of a smiling face; when the user clicks on the face it changes into a sad face. A variable represents the state of the game, it is initialized in the *setup()* method, and it can only be changed by the *mouseClicked()* P5 method. The *draw()* method simply looks at the value of the variable and decides which of the 2 images to draw on the canvas. As shown in Fig. 2, this is an instance of the model-view-controller design pattern and allows the 3 functions to work independently, allowing learners to smoothly approach game design and implementation and leverage on pen and paper design of the gameplay.



Data
variables
a
b
...
I remember ...
Controls + Rules
mouseClicked()
When PLAYER asks me*, I kick data guy!
Visualization
draw()
60 times per sec
I keep drawing what he says...
I never change any variable
* = the action only has effect if the variables are in certain states

Fig. 2. The game has three parts, reminiscent of the MVC design pattern.

This first exemplar was introduced in lecture 10 of 13, so after the students had worked with Scratch for the first 6 lectures, and after a few lectures about P5, events, graphics and animation. In that lecture, they were also presented with the recipe to create a generic point-and-click game. The goal of the lecture was to organize the students in groups of 2 to 3, customize and modify the games exemplars. After the class practiced with the idea of point-and-click games, we had a lecture structured as a *1-day game jam*. The students were again divided in groups, and this time they were encouraged to brainstorm, design and implement a point-and-click game. A moodboard was suggested as a way to quickly collect game-related images, to draw inspiration for the theme of the game, and eventually to use in the game implementation. The games created in this 1-day game jam were also the basis for the individual games that the students would deliver for their exam, together with a report. This short game jam was created to support the *generative* and *guided-discovery* scaffolding strategies from [11].

IV. Findings and Reflections

In order to evaluate our scaffolding approach to CT, we conducted a qualitative analysis and created an affinity diagram [17] on the assignments that the students have delivered for the *1-day game jam*. Our analysis focused on how the students related to the provided recipe to create point-and-click games. For the exam of the course, the students have to deliver a report with a discussion of two programs: one written in Scratch and the other in P5. They had the option to expand the games they created during the 1-day game jam and use them as the basis for the P5 part of the exam: 6 out of 10 took this option. The remaining 4 students implemented their own games or interactive animations, they showed confidence with JavaScript, previous knowledge of programming, and all got very high or top grades. The students who worked on the point-and-click games can be categorized as follows: those who followed our **recipe** customizing the images and rules, those who expanded the games by adding complex rules, many assets, and even multiple levels; and two students, who explored the *hidden object* genre, working with a large background image, drawing circles around the found items, and writing order-independent rules. Finally, many of the P5 programs delivered for the exam were well commented and showed a good use of functions as a mean to reduce code duplication and improve readability.

The main recurring problem in the point-and-click games was managing the complexity of having multiple levels. One student succeeded, and the resulting game was quite long to play, challenging and bug-free. Other attempts failed to resolve conflicts among the rules and the states of the images on screen. Some problems related to rules interfering with each other and the growing complexity of long point-and-click games, already manifested themselves during the lectures, allowing for deeper reflections during tasks supervision, and especially after the game jam. We also performed a simple analysis of the code submissions using JSHint, as visible in Table I; the first 6 rows are from students who worked with the point-and-click code.

TABLE I: ANALYSIS OF P5 CODE SUBMISSIONS (JSHINT).

No.	# functions	signature - largest	signature - median	# statements in a function - largest	# statements in a function - median	cyclomatic : largest	cyclomatic : median
1	5	4	0	48	20	18	5
4	5	4	0	235	33	116	5
5	7	5	0	25	3	20	1
6	5	4	0	101	26	43	5
7	7	5	0	25	3	20	1
8	7	5	0	26	3	20	1
2	12	2	0	28	6,5	7	1,5
3	7	0	0	12	7	5	4
9	4	0	0	51	6,5	21	1,5
10	8	0	0	17	4,5	5	1

The average "largest cyclomatic complexity" (next-to-last column in the table) of the point-and-click deliveries was 39.5, while the other games had 9.5. A higher cyclomatic complexity of the point-and-click games indicates that the students have implemented the logic of their games using many conditional statements, as suggested by our recipe, and that their games became much more complex than the examples we originally provided. In general, the students who worked with point-and-click code created more complex code, which is a positive outcome from a pedagogical point of view (also according to [10] and [6]). Interestingly, as the students appropriated the recipe and game examplars, they gained more opportunities for personal exploration of the game designs and coding, a very desirable development when supporting beginners (also according to [9]), as a resource for fostering further exploration of creative coding.

V. CONCLUSION

Our findings suggest that point-and-click games are an effective way to provide scaffolding and ease non-technical students into P5 programming, even in the face of severe limitation on the coding concepts that can be introduced. The game genre motivated our students by providing a recognizable spectrum of games; it also has enough expressive power and even allowed students to expand into similar domains, such as hidden object games. Adopting P5.js instead of plain JavaScript offered a smooth transition from Scratch to text-based programming, and the stability of P5's online IDE solved many of the typical problems related to beginner programmers working on different machines and systems. Our approach would also fit introductory programming courses in secondary education and for non-technical bachelor in other educational lines; for that, we are currently investigating porting our recipe to other coding environments adopted in Danish education.

REFERENCES

[1] J. M. Wing, "Computational thinking," Communications of the ACM, vol. 49(3), 2006, pp. 33-35.

[2] R. Chongtay, "Computational Thinking som redskab til problemløsning på tværs af fagområder," Computational Thinking. Teoretiske, empiriske og didaktiske perspektiver. N. Bonderup Dohn, R. Mitchell, and R. Chongtay (Eds.), Forfatterne og Samfundslitteratur, 2021, pp.123-146.

[3] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: blocks and beyond," Communications of the ACM, vol. 60 (6), June 2017, pp. 72–80. https://doi.org/10.1145/3015455

[4] A. Engin. Learn JavaScript with p5. js. Coding for visual learners. Springer, 2018.

[5] M. Tedre, and P. J. Denning, "The long quest for computational thinking," Proceedings of the 16th Koli Calling international conference on computing education research, 2016, pp. 120-129.

[6] F. Esteve-Mon, M., Llopis, and J. Adell-Segura, "Digital competence and computational thinking of student teachers," International Journal of Emerging Technologies in Learning (iJET), vol. 15 (2), 2020, pp. 29-41.

[7] P. J. Denning, and M. Tedre, "Computational thinking: A disciplinary perspective," Informatics in Education, vol 20 (1), 2021, pp. 361-390.

[8] Y. Li, A. H. Schoenfeld, A. A. diSessa et al. "Computational Thinking Is More about Thinking than Computing," Journal for STEM Education Research 3, 2020, pp. 1–18.

[9] C. Angeli, and M. Giannakos, "Computational thinking education: Issues and challenges," Computers in Human Behavior, vol. 105, 2020.

[10] N. Boblett, "Scaffolding: Defining the metaphor," Studies in Applied Linguistics and TESOL, 2012 vol. 12 (2).

[11] U. Kale, M. Akcaoglu, T. Cullen et al., "Computational What? Relating Computational Thinking to Teaching," TechTrends, vol. 62, 2018, pp. 574–584.

[12] D. A. Schön, Educating the reflective practitioner: Toward a new design for teaching and learning in the professions. Jossey-Bass, 2002.

[13] S. Papert, and I. Harel, "Situating constructionism," Constructionist Learning. I. Harel (Eds.), MIT Media Laboratory, vol. 36 (2), 1991, pp. 1-11.

[14] A. Valente, E. Marchetti and J. Wang, "Design of an educational multimedia library to teach Python to non-technical university students," 2020 9th International Congress on Advanced Applied Informatics (IIAI-AAI), Kitakyushu, Japan, 2020, pp. 169-175, doi: 10.1109/IIAI-AAI50415.2020.00041.

[15] A. Valente and E. Marchetti, "The road towards friendly, classroom-centered interactive digital contents authoring," 27th International Conference on Computers in Education´. Asia-Pacific Society for Computers in Education, 2019, pp. 38-46.

[16] A. Valente and E. Marchetti, "StickAndClick: sticking and composing simple games as a learning activity," Learning and Collaboration Technologies. Human and Technology Ecosystems, P. Zaphiris, & A. Ioannou (Eds.), the 22nd HCI International Conference, HCII 2020, Proceedings, 2020, vol. 2, pp. 333-352. Springer. Lecture Notes in Computer Science Vol. 12206. https://doi.org/10.1007/978-3-030-50506-6_24

[17] J. Preece, Y. Rogers, and H. Sharp, Interaction design: beyond human-computer interaction, John Wiley & Sons, 2023