CBIE 2023
CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO
Uma escola para o futuro: Tecnologia e conectividade a serviço da educação

Introducing programming logic consistently through gamification

João Marcelo Borovina Josko, Francisco de Assis Zampirolli

[1]Federal University of ABC (UFABC)
Av. dos Estados, 5001 – Santo André – 09210-580 – SP – Brazil

{marcelo.josko, fzampirolli}@ufabc.edu.br

Abstract. *Introducing programming logic to students with varying backgrounds is challenging for different reasons. To address this challenge, tutors may use various approaches, including computer games. Although there is a vast literature on this topic, few studies focus on accessible environments that allow students to run and share code. Hence, we conducted a small case study on one online group of CS0 to capture (using a survey) students' perspectives ($N = 48$) on our teaching method that introduces programming logic through gamification on Google Colab. Our quantitative analyses suggested that using games positively affected students' motivation on CS0 topics. Moreover, the average failure rate of CS0 groups (between 2020 and 2021) was higher than the one using our proposed method (*26.2% *and* 17.4%*, respectively).*

1. Introduction

In today's era of digitalization and technological advances, possessing computer skills is crucial. To equip students from various fields with computational skills and programming logic, universities worldwide offer introductory courses even for students not directly related to computing, such as mathematics or engineering [Resnick and Rusk 2020].

Several studies have applied game thematic to introductory programming CS0 and CS1. For example, [Stephan et al. 2020] presented a multimedia game to engage students in learning programming concepts, while others adopted robot fights for the same purpose [Lapeña et al. 2022]. Moreover, [Koppurapu 2021] used interactive development platforms (Jupyter Notebook) to introduce programming concepts while students built a board game.

Although digital games have proven valuable in teaching programming concepts, few studies have explored game ideas with independent platforms like Google Colab. Only [Koppurapu 2021] has been identified to adopt Jupyter Notebook to teach computational thinking to K-12 students.

To address this gap in higher education, we conducted a case study to examine how learners perceive our pedagogical intervention, which incorporates gamification elements during a real-life CSO group ($N = 48$). We used a survey to collect data and conducted quantitative and qualitative (composite narrative) analysis procedures. This work aims to answer the following research question:

> *Which elements of our pedagogical intervention (e.g., gamification, automatic feedback) do students perceive as relevant to their learning?*

The present work has the following structure: In Sections 2 and 3, we discuss concepts of digital games and outline related works. Next, we sketch our pedagogical approach in Section 4, whose results and the corresponding discussion we discuss in Sections 5 and 6, respectively. Finally, we list some threats to validity (Section 7) and conclude this work (Section 8).

2. Background

A *digital game* is a rule-based system that rewards players for performing actions. These actions accumulate as players work to overcome challenges and achieve specific goals. Because of these characteristics, combined with an engaging narrative and visual design, digital games have the potential to engage players fully [Plass et al. 2020]. Engagement is crucial for motivating players to voluntarily persist in performing a task. This property of games is especially relevant in educational settings where students need to learn specific topics [Plass et al. 2020].

There are three main approaches to introducing game dynamics in pedagogical interventions: *game-based learning*, *playful learning*, and *gamification*. Game-based learning involves completely reformulating learning activities using game properties and educational theories. On the other hand, playful learning combines game dynamics with other resources, such as pedagogical techniques, to make learning activities enjoyable. Finally, gamification involves adding game elements like rewards and rankings to traditional learning activities.

The present work uses gamification to introduce essential elements of digital games, such as object movement and collision detection, in programming logic activities. The goal is not to develop complex games but to leverage game elements to enhance learning.

3. Related Works

The literature on using digital games in computer programming education is extensive, but we focus mainly on works that teach programming using gamification at the undergraduate level.

Several works reported the benefits of playing or developing games in computer programming teaching. For example, some studies have discussed the positive effects of gamification on students in programming learning [Marín et al. 2018], attitude towards learning the subject [Domínguez et al. 2013], or collaboration in remote sessions [Knutas et al. 2014]. In contrast, other studies found that games moderately impacted motivation [Rodrigues et al. 2021] and achievement [Chang et al. 2020].

Regarding playing a game to learn, the literature has discussed several games using different mechanics or dynamics elements. For example, [Stephan et al. 2020] presented a four-stage game based on CodeBlock and multimedia components (SFML library) to teach programming concepts, named GameProgJF. CoMA [Bachtiar et al. 2018] is a web-based game that uses a leaderboard, challenges, and allows students to compete against colleagues during Java learning. In another way, several studies have used the RoboCode game environment to engage students in producing robot tanks to fight each other on a battlefield [Lapeña et al. 2022].

In addition to creating "mini-worlds," few studies have explored the potential of interactive development platforms incorporating game elements to teach programming concepts. For example, only one study, [Koppurapu 2021], has utilized a Jupyter Notebook for students to implement a board game as the final product of a computational thinking course.

Finally, other works have adopted game ideas to teach programming beyond playing or building digital games. For example, [Llerena-Izquierdo and Sherry 2022] used the escape room theme to reinforce learning in Python. These rooms were created in Google Forms and combined videos, QR codes, and questions about Python topics that students must interact with to complete a sentence and leave the corresponding room.

Our work builds on previous literature using the Google Colab platform, which is easy to access and independent of any specific programming environment. In addition, we use gamification to teach fundamental programming concepts. Finally, by adopting this approach, we aim to engage students and improve their understanding of programming logic.

4. Method

This section presents the context of the CS0 course (Section 4.1) and our pedagogical intervention (Section 4.2), focusing in how we used the `Pygame` library as a motivational factor. Finally, we discuss our data collection and analysis procedures in Sections 4.3 and 4.4.

4.1. Our Context

Programming Logic (CS0) is a mandatory course for all students in the first academic term of our university's interdisciplinary science and technology program. As a result, students with varying levels of programming experience are expected to enrol. The course spans 12 weeks, with two hours of practical classes each week. However, due to the COVID-19 pandemic, the 2020-2021 cohort attended the CS0 course entirely online. Professors gave students synchronous or asynchronous classes through the Moodle Learning Management System. For the September-December 2021 term, we offered a course combining synchronous and recorded classes.

4.2. Pedagogical Intervention

Table 1 presents the subject syllabus for our CS0 course, which has been informed by our previous work [Zampirolli et al. 2021b]. We designed the first five weeks of the course to help students become gradually comfortable writing and executing simple code that produces meaningful results, focusing each week on a specific topic. This approach prepares students for the more formalized algorithmic thinking introduced in the sixth week and helps reduce anxiety for those who have never programmed.

Table 1. CS0 syllabus

Week	Content
1	Development Environments
2	Databases with Pandas Python Library
3	Graphs of Functions
4	Descriptive Statistics
5	Correlation and Linear Regression
6	Sequential Programming
7	Conditional Programming
8	Loops
9	Modeling and Simulation 1
10	Modeling and Simulation 2
11	Review
12	Final Exam

To enhance student engagement and facilitate discussion of topics covered in weeks 6 to 10 (Sections 4.2.1 to 4.2.4), we have introduced gamification elements in our current iteration of the course. Throughout the course, we also use Google Colab and explanatory videos to present technical terminology in a language that is more accessible to students, reducing potential language barriers and promoting inclusivity in the classroom.

The course assessment strategy consists of five parametrized tasks (40% of the final grade) focusing on the latest topic discussed, which students receive before each class. These submissions are automatically graded using the integration of MCTest, Moodle, and VPL. In addition to these tasks, there is a final exam (40% of the final grade) and a final project (20% of the final grade). The final project requires students to explain the code of one of the games available on Program Arcade Games.

4.2.1. Sequential Programming

We utilize gamification to introduce fundamental programming concepts gradually, i.e., one by one. This approach allows students to observe how these concepts combine to form a simple but complete game by week 10, preventing them from feeling disoriented. In week 6, we introduce the sequential structure by discussing the first part of the game, as illustrated in Figure 1.

```
1  #%%writefile jogo00.py
2  import pygame, os
3  from _jogo import *
4  pygame.init()
5  os.environ["SDL_VIDEODRIVER"] = "dummy"
6  screen = pygame.display.set_mode((jogo.WIDTH,jogo.HEIGHT))
7  screen.fill(jogo.GRAY)
8  jogo.drawScreen(screen,2)
```

Figure 1. Game part one - Showing its screen using sequential structure.

The previous weeks prepared students to understand the first part of the game: showing the game screen. Such an approach allowed us to focus on the changes caused by the order of steps of sequential structure and relate to the previous concepts. In weeks 1 and 2, we introduced the meanings associated with importing library (lines 2−3) and the notion of variable (line 6). This discussion was supported by a tiny program where students had to import the math lib, use one of its functions and store its return value. Next, weeks 3 to 5 focused on manipulating different functions (e.g., plotting simple graphs) with some variation of parameters (lines 4,6−8).

We omitted the details of the `jogo` library (Figures 2 and 3) from the students, but we replicate here the code for use by the community. The `jogo` class sets up some game characteristics (e.g., predefined colors). In contrast, the `drawScreenColab` method is responsible to turns the game screen into an image and displaying it in Colab (as a screenshot of a hidden GUI).

```
1  import cv2, sys, pygame, time, pandas as pd
2
3  class jogo(object):
4    IN_COLAB = 'google.colab' in sys.modules
5    GRAY = (240, 240, 240)
6    RED = (255, 0, 0)
7    GREEN = (0, 255, 0)
8    BLUE = (0, 0, 255)
9    BLACK = (0, 0, 0)
10   WHITE = (240, 255, 255)
11   WIDTH, HEIGHT = 800, 400
12
```

Figure 2. The `jogo` library, defined in the file `_jogos.py`

```
13  def drawScreenColab(screen):
14      from google.colab.patches import cv2_imshow
15      view = pygame.surfarray.array3d(screen)
16      view = view.transpose([1, 0, 2])
17      img_bgr = cv2.cvtColor(view, cv2.COLOR_RGB2BGR)
18      cv2_imshow(img_bgr)
19
```

Figure 3. `drawScreenColab` method of the `jogo` library

4.2.2. Conditional Programming

Google Colab has limitations on running codes based on `Pygame`, including the impossibility of controlling game elements. We used this limitation to introduce a problem to the students in the seventh week: How could our game be automatically set to run on our local computer (to get more interactivity) or on the Web?

This actual scenario was the basis for introducing the conditional idea and purpose. First, we used visual materials (diagrams) to exemplify the behaviour of the if statement using different examples of increased complexity. Then, founded on this understanding, we presented the if statement per se, reusing the previous examples.

Finally, we returned to the opening question and discussed the evolved `drawScreen` method with students, as illustrated in Figure 4. This method exemplifies conditional game usage on lines 22,25,28,29,31, and 35. Furthermore, we left on the Colab detailed instructions to students on downloading the source code, installing Python and `Pygame` and running the code.

```
20  def drawScreen(screen, tempo=0):
21      pygame.display.update()
22      if jogo.IN_COLAB: # run in Colab
23          from google.colab import output
24          jogo.drawScreenColab(screen)
25          if tempo:
26              time.sleep(tempo)
27              output.clear()
28      else: # run in console
29          if tempo>0:
30              time.sleep(tempo) # python v3.8
31          else:
32              run = True
33              while run:
34                  for event in pygame.event.get():
35                      if event.type == pygame.QUIT:
36                          run = False
37
```

Figure 4. `drawScreenColab` method with conditionals statements

4.2.3. Loops

To introduce the looping idea, we asked students how to move a rectangle (line 10) on the screen game. Similar to the previous topic, we used visual aids and increasing complexity examples to discuss the loop behavior. Next, we returned to our main program (Figure 1) and

made gradual improvements to create the version shown in Figure 5 [1]. We started by having the rectangle glide smoothly to the right until it collided with the right edge of the screen, changing its color to blue (lines $16-17$). Then, we had it smoothly glide to the left (line 19) until it touched the left edge, with this process repeating 25 times as defined in line 11.

```
1  #%%writefile jogo01.py
2  import pygame, os, sys
3  from _jogo import *
4  pygame.init()
5  os.environ["SDL_VIDEODRIVER"] = "dummy"
6  screen = pygame.display.set_mode((jogo.WIDTH, jogo.HEIGHT))
7  pygame.display.update()
8  LEFT, RIGHT = 7, 3 # directions
9  SIZE, MOVESPEED = 50, 50 # size and speed of rectangle b1
10 b1 = {'rect':pygame.Rect(600, 200, SIZE, SIZE), 'color':jogo.RED, 'dir':RIGHT}
11 for i in range(25):
12   screen.fill(jogo.GRAY)
13   b1['rect'].left += MOVESPEED
14   pygame.draw.rect(screen, b1['color'], b1['rect'])
15   jogo.drawScreen(screen,0.5)
16   if b1['dir'] == RIGHT:  # touch right/left?
17     if b1['rect'].right > jogo.WIDTH - SIZE:
18       MOVESPEED *= -1
19       b1['dir'], b1['color'] = LEFT, jogo.BLUE
20   if b1['dir'] == LEFT:
21     if b1['rect'].right < SIZE:
22       MOVESPEED *= -1
23       b1['dir'], b1['color'] = RIGHT, jogo.BLACK
24 jogo.drawScreen(screen)
```

Figure 5. Code to illustrate the movement of rectangle to introduce loop.

4.2.4. Modeling and Simulation

During our modeling and simulation classes, we concluded our game that simulates the movement of a car and an asteroid in a two-dimensional space. We introduced methods to detect collisions in two-dimensional spaces to simulate the game. When the game is run in Colab, the car moves randomly left or right until it touches an edge or collides with the asteroid. If the game is run in the computer's console, the car's movements are controlled using the arrow keys. All four Colab files for the game are available here.

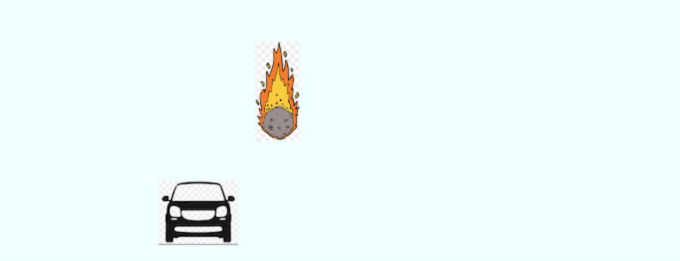

Figure 6. Final game GUI

4.3. Data Collection

We conducted two voluntary surveys to gather data from students in our programming course. The first survey was administered at the beginning of the course and collected

[1] Adapted from dainf.ct.utfpr.edu.br/petcoce/wp-content/uploads/2013/09/document.pdf.

demographic information and academic interests related to learning programming. We explained the purpose of the research to the students, emphasized that participation was voluntary, and outlined our procedures for protecting their privacy. The second survey was administered before the final exam and followed a similar structure to the survey used by [Zampirolli et al. 2021b]. It included 11 Likert-based questions and one open-ended question to gather feedback on the course content and strategies.

4.4. Analysis Methods

To analyze the data from surveys, we followed a four-step process. First, we organized the data into a manageable format that allowed us to perform further analysis. In the second step, we coded the few open-ended answers using inductive coding. Next, we explored the Likert answers to determine which statistical methods would be more appropriate for analyzing the data: parametric or non-parametric. Finally, in the last step, we applied the chosen methods to the Likert answers.

5. Results

In the first survey, we asked students if they would prefer to learn programming by developing a game. Out of the 48 students who responded, 45 of them (94%) indicated a preference for this approach. However, three students (6%) reported that they preferred a different approach but did not provide any alternative suggestions. When asked about their learning goals for the course, most students (73%) indicated that they hoped to learn the programming language and better understand the technology around them. A smaller percentage of students indicated they had no prior knowledge of the subject (7%) or wanted to practice programming logic (20%).

The second survey received only 32 responses (67%) for the Likert questions (Table 2). We analyzed these responses and found that most of the questions resemble a normal distribution, i.e., skewness or kurtosis between -2.575 and 2.575 for 99% confidence level (CL) [Ryan 2013], except for $\mathcal{Q}14$ and $\mathcal{Q}15$. Therefore, we used a non-parametric method ($Wilcoxon\ Signed-Rank$) for these two questions and an equivalent parametric method ($t-test$) for the remaining. Next, we applied Cronbach's Alpha and obtained .86, indicating that Likert data had good internal consistency. Finally, we tested the hypothesis below.

H_0: all approaches had a neutral effect on students' learning, $mdn = 3$.
H_1: all approaches positively affected students' learning, $mdn > 3$.

Table 3 displays the results of a one-group $t-test$ measuring students' perception of the value of our learning approaches. The findings indicate that most questions differ from the null hypothesis (H_0). For example, students' preference for our library ($\mathcal{Q}3$) was highly regarded, with a median score of 4, $t(31) = 3.30$, $p < 0.01$, ES=moderate (.57). In contrast, the assistant ($\mathcal{Q}10$) did not receive as positive a response, with a median score of 4, $t(31) = 2.38$, $p > 0.01$, ES=small (.41).

Similarly, students perceived automatic correction and feedback as helpful, as indicated in Table 4. For example, the results for the first approach suggest that the alternative hypothesis (H_1) is statistically significant by the one group $Wilcoxon\ Signed-Rank$ test, $N = 31$, T=7.60, $p < 0.01$, ES=large (.88).

Finally, 11 students ($\approx 34\%$) answered the open-ended question with an average length of 186 words (STD=69). Most students ($n = 10$) requested more accessible weekly checklists or PE resolutions during synchronous lessons. One student left a message of gratitude.

Id	Question
9	Regarding the sequence of subjects in Table 1, do you think it was adequate for CS0?
10	Were assistants helpful in teaching/learning (e.g., lists of exercises, theory, etc.)?
12	Do you agree with using the same PE for all classes, which leads to LEVELING evaluations and apprenticeship (e.g. by lists on Moodle), however, without suppressing the professor's autonomy in teaching methodologies?
14	Do you think that the AUTOMATICAL CORRECTIONS of the Moodle exercises, which gave marks automatically, were constructive in learning?
15	Concerning FEEDBACK (Python error messages that arise when assessing an activity) of the exercises on Moodle, do you think it helps the learning process?
17	Considering the INDIVIDUALIZED LISTS OF EXERCISES, with slightly different statements and in shuffled order (emailed to the students), do you think they helped DIMINISH PLAGIARISM in the course and therefore contributed to the learning process through FAIRER evaluations?
18	Do you think drawing questions from a larger QB reduces plagiarism? For example, for a list of five multiple–choice questions, some works assert that one should draw them from a QB with at least 25 distinct items (see `tandfonline.com/doi/abs/10.1080/02602930020022273`).
19	What is your GENERAL evaluation of CS0 concerning GENERAL TEACHING resources employed in the lectures (Slides, Notebooks, Videos, etc.)?
23	What is your GENERAL evaluation of CS0 for resources employed in ASSESSMENTS (Moodle, Lists, Automatic Assessment, etc.)?
24	What is your OVERALL assessment of using the `Pygame` library as a MOTIVATION to learn programming logic?

Table 2. Post-class Likert questions

Table 3. $t-test$ **analysis for all questions, except** $\mathcal{Q}$**14 and** $\mathcal{Q}$**15**

$\mathcal{Q}$	t	df	$p-value$	Mean	Std. Deviation	Median	Effect Size (ES)	Power (CL=99%)
$\mathcal{Q}$9	6.73	31	$<.01$	4.2	.9	5	.86	.99
$\mathcal{Q}$10	2.38	31	.012	3.6	1.41	4	.41	.47
$\mathcal{Q}$12	7.92	31	$<.01$	4.2	.87	4	.86	.99
$\mathcal{Q}$17	8.91	31	$<.01$	4.3	.81	4	.86	1
$\mathcal{Q}$18	3.75	31	$<.01$	3.75	1.24	4	.59	.82
$\mathcal{Q}$19	7.72	31	$<.01$	4.25	.91	5	.86	.99
$\mathcal{Q}$23	7.60	31	$<.01$	4.2	.9	4	.86	.99
$\mathcal{Q}$24	3.30	31	$<.01$	3.7	1.17	4	.57	.79

Table 4. $Wilcoxon\ Signed-Rank$ **analysis for questions** $\mathcal{Q}$**14 and** $\mathcal{Q}$**15**

$\mathcal{Q}$	T	$Median$	$z-score$	Effect Size (ES)	Power (CL=99%)
$\mathcal{Q}$14	7.60	5	$<.01$	.88	.99
$\mathcal{Q}$15	3.30	4	$<.01$	.59	.79

6. Discussions

Our study suggests that incorporating gamification and automatic correction feedback into CS0 courses can improve students' motivation toward building their programming understanding. Our gamification findings are consistent with the results of [Marín et al. 2018, Domínguez et al. 2013], who reported that gamification is an effective way to motivate students to keep studying and learning programming logic. Moreover, automatic correction and feedback seemed to help students by contrasting their actual performance against the required one, in line with previous studies [Araujo et al. 2021],

including ours [Zampirolli et al. 2021b, Zampirolli et al. 2021a, Borovina Josko 2021].

Over ten years, from 2009 to 2019, our university offered a total of 734 groups of the CS0 course, with an overall average failure rate of 24.8%. However, during the pandemic years of 2020 and 2021, 111 of these groups were conducted under quarantine conditions, resulting in a higher average failure rate of 26.2%. To address this issue, we introduced a motivational method that uses games as part of our pedagogical intervention. As a result, the failure rate was lower in a group that used this intervention, at 17.4%.

These results suggest that incorporating games into the curriculum can help improve student engagement and performance during challenging times. However, the sample size of the group that received this intervention was small, and further studies with larger sample sizes are needed to fully assess this approach's effectiveness.

7. Threats to Validity

Historically, CS0 courses at our university have had students with diverse programming backgrounds. However, in this case study, we could not determine the variation among the students who attended our group compared to others. Therefore, it is likely that our study's participants do not fully represent the population of students being studied.

8. Conclusions

This study examines the initial impacts of our teaching approach, which involved gradually building a simple game and incorporating automatic grading and feedback to introduce programming logic. The gaming approach was based on the Python programming language's `Pygame` library, and it could be run on a computer console or simulated in Google Colab. In addition, we analyzed the feedback of 48 participants who reported that our approach effectively supported and motivated their learning process.

While the method proposed in this article has received positive feedback, further research is needed to draw more conclusive results. Specifically, a comparative study involving control classes would allow for a more comprehensive analysis of students' ability to apply their acquired knowledge.

Data Availability Statement

The data supporting the findings of this study are available here.

References

Araujo, L. G. J., Bittencourt, R. A., and Chavez, C. v. F. G. (2021). Python enhanced error feedback: Uma ide online de apoio ao processo de ensino-aprendizagem em programação. In *Anais do Simpósio Brasileiro de Educação em Computação*, pages 326–333. SBC.

Bachtiar, F. A., Pradana, F., Priyambadha, B., and Bastari, D. I. (2018). Coma: Development of gamification-based e-learning. In *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–6. IEEE.

Borovina Josko, J. M. (2021). Mixing cognitive and affective approaches in teaching introductory programming. ITiCSE '21, page 641, New York, NY, USA. Association for Computing Machinery.

Chang, C.-S., Chung, C.-H., and Chang, J. A. (2020). Influence of problem-based learning games on effective computer programming learning in higher education. *Educational Technology Research and Development*, 68(5):2615–2634.

Domínguez, A., Saenz-de Navarrete, J., De-Marcos, L., Fernández-Sanz, L., Pagés, C., and Martínez-Herráiz, J.-J. (2013). Gamifying learning experiences: Practical implications and outcomes. *Computers & education*, 63:380–392.

Knutas, A., Ikonen, J., Nikula, U., and Porras, J. (2014). Increasing collaborative communications in a programming course with gamification: a case study. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*, pages 370–377.

Koppurapu, P. R. (2021). K-12 teacher computer science education: computational thinking curriculum centered around data science. Master's thesis, California State University, Sacramento.

Lapeña, A., Marín, B., and Vos, T. (2022). Gipppy: Game for introductory programming practice in python. In *INTED2022 Proceedings*, pages 5598–5606. IATED.

Llerena-Izquierdo, J. and Sherry, L.-L. (2022). Combining escape rooms and google forms to reinforce python programming learning. In *Communication, Smart Technologies and Innovation for Society*, pages 107–116, Singapore. Springer Singapore.

Marín, B., Frez, J., Cruz-Lemus, J., and Genero, M. (2018). An empirical investigation on the benefits of gamification in programming courses. *ACM Transactions on Computing Education (TOCE)*, 19(1):1–22.

Plass, J. L., Homer, B. D., Mayer, R. E., and Kinzer, C. K. (2020). Theoretical foundations of game-based and playful learning.

Resnick, M. and Rusk, N. (2020). Coding at a crossroads. *Communications of the ACM*, 63(11):120–127.

Rodrigues, L., Toda, A. M., Oliveira, W., Palomino, P. T., Avila-Santos, A. P., and Isotani, S. (2021). Gamification works, but how and to whom? an experimental study in the context of programming lessons. In *Proceedings of the 52nd ACM technical symposium on computer science education*, pages 184–190.

Ryan, T. P. (2013). *Sample size determination and power*. John Wiley & Sons.

Stephan, J., Oliveira, A., and Renhe, M. C. (2020). O uso de jogos para apoiar o ensino e aprendizagem de programação. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 381–390, Porto Alegre, RS, Brasil. SBC.

Zampirolli, F. A., Borovina Josko, J. M., Venero, M. L. F., Kobayashi, G., Fraga, F., Goya, D., and Savegnago, H. (2021a). An experience of automated assessment in a large-scale introduction programming course. *Computer Applications in Engineering Education*.

Zampirolli, F. A., Sato, C. M., Savegnago, H. R., Batista, V. R., and Kobayashi, G. (2021b). Automated assessment of parametric programming in a large-scale course. In *2021 XVI Latin American Conference on Learning Technologies (LACLO)*, pages 357–363.