Programming Unplugged: A Computer-less Introduction to Programming

AMY LARSON

Mathematics, Statistics and Computer Science
Augsburg University
Minneapolis, MN 55454
larsonam@augsburg.edu

ABSTRACT

Programming Unplugged [4] is a series of assignments designed to build students' computational skills and acclimate them to rigid syntax. Students read and write simple code to create patterns on a grid with colored tiles – not in simulation, but with real physical elements. In the first assignment, students read code to place tiles (e.g. pink.place(2,3)). The code uses variables, randomization generated from a die (e.g. x=roll()) or random tile selection (e.g. tile=random()), if-statements and while loops. Students are introduced to the language with simple code examples that they *execute*. Once comfortable with reading code, students choose from a collection of coding problems described with a pattern image and code requirements (e.g. start the pattern at a random position). Students share code with their peers for testing, giving students practice executing and debugging code. In the second week, the patterns and constraints increase and require more complex code structures. In the third week, students are introduced to while loops to meet constraints such as "place tiles in a row until the edge of the board is reached." Students demonstrated high engagement during these activities, and it allowed for an easy entry point into programming. It is not yet known if it is a better introduction to programming than immediately starting with Python.

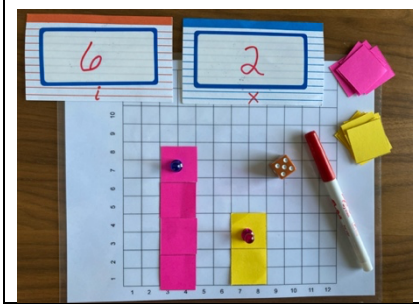
1. INTRODUCTION

Unplugged is a pedagogy common in K-12 education to teach computational thinking without a computer [1]. The success as a pedagogy for computer science is context-dependent and does not necessarily advance students faster than those who start on the computer. However, it has been shown to increase self-efficacy and encourage exploration after transitioning to coding on a computer [3]. Furthermore, it can be useful for resource-constrained classrooms with limited computer access. At Augsburg, while our classrooms are not resource-constrained, many of our students have had limited exposure to computers, to mathematics and even less to programming when first entering the classroom. There are many hurdles on their path to success. Furthermore, students significantly struggle with abstraction and the rigid syntax of programming. The assignment series *Programming Unplugged* is an attempt to bring programming into the physical world tying motion to code execution, and to separate computational thinking from syntax. Additionally, it delays the need to learn how to manage the computer. In the course, students worked unplugged for the first 3 weeks before transitioning to Python.

2. OVERVIEW OF ASSIGNMENTS

In pairs, students received a kit that includes a 12x12 grid, 2x2 colored tiles, a die, 2 game pieces and dry-erase cards to store variables. A language was created that manipulates tiles to create various patterns (*see Figure 1).* In the left image in Figure 1, a histogram has been constructed from a coding problem that repeatedly selects a random tile either a fixed number of times (counting loop) or until the top of the grid is reached. Coding problems are created from the combination of two cards: an image card with a tile pattern and a text card that further constrains the requirements, for example randomizing the starting

position on the grid or the order that the tiles are placed (*see Figure 2*). Notice in the image cards (top row of Figure 2) that there is an Image A and Image B on each card and the text card (bottom row of Figure 2) refers to one of those images. For some image cards, A and B are identical so that the text requirement cards can be paired with any of the image cards. Separating the images and constraints means they can be recombined in many ways to create a large pool of coding problems that students can choose from.

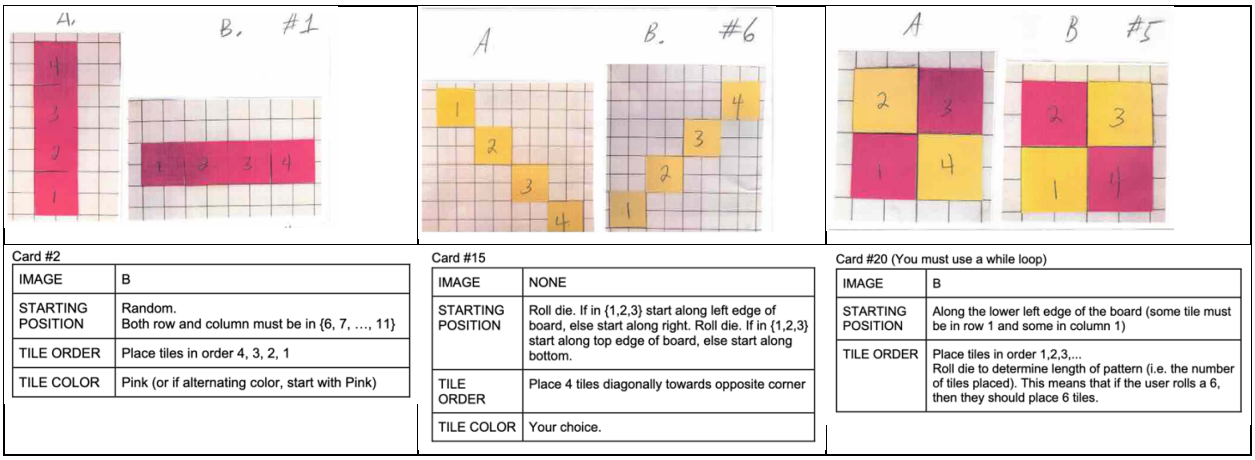


FUNCTIONS / OPERATIONS

tile.place(x,y) - example: Pink.place(1,5)
marker.place(x,y) - example: L1.place(2,12)
tile.place(marker) - example: Pink.place(L1)
marker.shift(direction,units) - example: L1.shift(UP,1)
roll() - physically roll the die to get a random number
random() - physically grab tile (without looking) to get random tile color
print(value) - displays value on a piece of paper. example: print(x)

Figure 1. On the left is the *Programming Unplugged* kit consisting of a 12x12 grid, 2x2 tiles, game pieces (position markers), and dry-erase "variable" cards. On the right is the corresponding programming language.

To begin *Unplugged*, students receive code that they *execute* to produce patterns on the grid. Students have to physically roll die and place tiles. Next, they receive coding problems by randomly selecting an image card and a text requirements card, as seen in Figure 2. They write the code (on paper with a pencil!), then share code with others who debug it by executing the code and building the pattern. Humans are much more forgiving with respect to missing parentheses and misaligned code, so although we focus on syntax, the real emphasis is on their problem-solving skills. Later in the semester, a Python version of the activity was created for students to experiment with.



Card #2

IMAGE	B
STARTING POSITION	Random. Both row and column must be in {6, 7, …, 11}
TILE ORDER	Place tiles in order 4, 3, 2, 1
TILE COLOR	Pink (or if alternating color, start with Pink)

Card #15

IMAGE	NONE
STARTING POSITION	Roll die. If in {1,2,3} start along left edge of board, else start along right. Roll die. If in {1,2,3} start along top edge of board, else start along bottom.
TILE ORDER	Place 4 tiles diagonally towards opposite corner
TILE COLOR	Your choice.

Card #20 (You must use a while loop)

IMAGE	B
STARTING POSITION	Along the lower left edge of the board (some tile must be in row 1 and some in column 1)
TILE ORDER	Place tiles in order 1,2,3,... Roll die to determine length of pattern (i.e. the number of tiles placed). This means that if the user rolls a 6, then they should place 6 tiles.

Figure 2. The top row has samples of pattern cards that students use as requirements. The bottom row has samples of text requirement cards that are paired with the pattern cards to specify a coding problem. The requirements progress in difficulty from no if-statements to a combination of loops and if-statements.

3. DISCUSSION

In the spirit of *Nifty Assignments* of ACM's SIGCSE (Special Interest Group in Computer Science Education) documented at http://nifty.stanford.edu/, the assignment will be discussed with respect to audience, difficulty, topics, strengths, weaknesses and variants.

Audience

This assignment series is an introduction to programming for students with no previous experience. The unplugged version is probably not appropriate for students with previous experience; however, the same concepts could be applied to the Python version, providing students with a visualization of their code which could aid in learning to debug.

Difficulty

The difficulty is at the discretion of the instructor, but the intended audience is students new to programming. It can focus on simple statements using variables or sophisticated use of while loops and if-statements. Additionally, the language can be extended to bring in more complex language structures.

Topics

As an introduction, it focuses on computational thinking and basic language structures such as variables, functions, if-statements and loops. A die and random tile selection emphasize the need for variables and if-statements.

Strengths

The activity had high engagement from students. There was a lot of energy and active group discussion in the room as they grappled with the problems. And there was physical movement at the table as they rolled dice and placed tiles, as well as around the room as they went to the table to get more problems or to share code with other groups. Students were also able to relate step-by-step instructions with physical movement – they embodied code execution. Another strength is that it removed the rigid nature of syntax that can consume students for hours over simple omissions or errors. Struggling with syntax is a barrier to students learning in how to think computationally to solve problems.

Another strength is that it required no additional equipment to engage in the activity. Students in their first semester of their first year are struggling to get oriented, so getting a textbook and computer into everyone's hand within the first week of classes is a challenge. This activity means no one gets behind because they did not or cannot get the textbook. It should also level the playing field between those who have computer experience and those who do not. Students do not have to figure out how to download a file and run it in Python. They do not have to figure out how to install Python on day one.

Weaknesses

With respect to student learning, I made the observation that some students memorize the coding structures without understanding the meaning or purpose of the code. For example, in an exam I saw reference to rolling a die or placing a tile, even though the problem is not related to the *Unplugged* problems. While I do not think this is unique to this series of assignments, I think it might more strongly reinforce this behavior because it is their introduction to these concepts and the solutions are all very similar. Consequently, I think when students saw a reference to a counting loop later in the semester, their brain retrieved the loop structures for *Unplugged* despite the fact that the problem was unrelated.

There are logistical problems as well. Constructing the kits is a practical concern for large classes. This might be resolved with a trip to your favorite print shop, but it is probably not an easy undertaking for classes much larger than 40 students (1 kit per 2 students was sufficient).

Variants

The language can be extended or modified. The problems and patterns can be changed, for example, creating shapes like pyramids or rectangles. Additionally, this can become a "plugged" version - the code has been provided on the Github site [4]. (Warning, this is not good code – it was quickly constructed to closely resemble the *Unplugged* version and still be accessible to students with little programming experience. There is much room for improvement there.)

Assessment

Assessment of student work for the *Unplugged* assignments is no different than it is for other programming assignments. Code sharing provided formative feedback and exam questions were in the Unplugged language.

4. CONCLUSION

Programming Unplugged was used for the first time in the current semester Spring 22. As stated above, students were highly engaged in the activity and I think it provided an easy entry point into programming. Unfortunately, this level of engagement was not sustained as we transitioned to coding. As (should have been) anticipated, once students came up against the predictable frustration with syntax and intellectual challenge of harder problems, some stopped engaging in the course. Overall, I think there is much promise in this series of assignments, and its development will continue. There has not been formal assessment of this as a pedagogy. Ideally, it would be good to compare the outcomes of students who start the course with this assignment versus those that immediately launch into Python. As the assignment is refined, we will also consider methods to assess the use of this and any other unplugged activity.

REFERENCES

[1] T. Bell. CS unplugged or coding classes? Communications of ACM 64, 5 (May 2021), pp 25-27. DOI:https://doi.org/10.1145/3457195

[2] L. Busuttil and M. Formosa. Teaching Computing without Computers: Unplugged Computing as a Pedagogical Strategy (2021). *Informatics Educ., 19*, 569-587.

[3] Felienne Hermans and Efthimia Aialoglou. To Scratch or not to Scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17). Association for Computing Machinery, New York, NY pp 46-56. DOI:https://doi.org/10.1145/3137065.3137072

[4] Amy Larson. *Programming Unplugged* resources. https://github.com/lars1050/ProgrammingUnplugged