FOR PLAY: LEARN TO CODE FOR LOOPS THROUGH PLAY

Learn to Code For Loops Through Play
DURELL BOUCHARD

Abstract

Many students drop out of computer science programs, in part, because they find the work to be tedious and boring. Incorporating play into the learning process can help make learning how to program more fun. To bring the joy of programming to more students, I present the educational browser game *For Play*. The game helps players learn to write code using several common programming patterns that use for loops, a common programming construct that facilitates executing repeated code. *For Play* is designed using research from game design to make it fun and engaging and research from computer science pedagogy to improve learning outcomes.

For Play: Learn to Code For Loops Through Play

Many of the students that enter college intending to major in computer science ultimately graduate with a different major. One reason for this is that not all students find learning computer science enjoyable. Biggers, Brauer, and Yilmaz found that students who dropped out of a computer science program reported that one reason was that the work was tedious and boring (2008). Games, in contrast, are fun. So, by incorporating educational programming games into computer science curricula, learning to program can be more fun.

Duckworth (2016) argues that the motivation required to develop skill at a task is engendered by having a deep interest in it. And an interest is cultivated when the task is initially fun. So, if more students experience the fun and joy of programming early in their education, they will be more likely to develop an interest that will motivate them to persevere through the less fun deliberate practice (Ericsson, Krampe, & Tesch-Römer, 1993) required to become proficient at programming and obtain a degree in computer science.

I have created a game, For Play, that helps students learn to write programs that utilize for loops. A for loop is a programming control structure that facilitates creating repeating code. They can simplify the solutions to complex problems and are a fundamental part of any sophisticated program that deals with large amounts of data. While for loops are themselves a simple concept, their use can be quite complicated when paired with other concepts such as variable updating. This makes for loops an excellent choice for deliberate practice in a game.

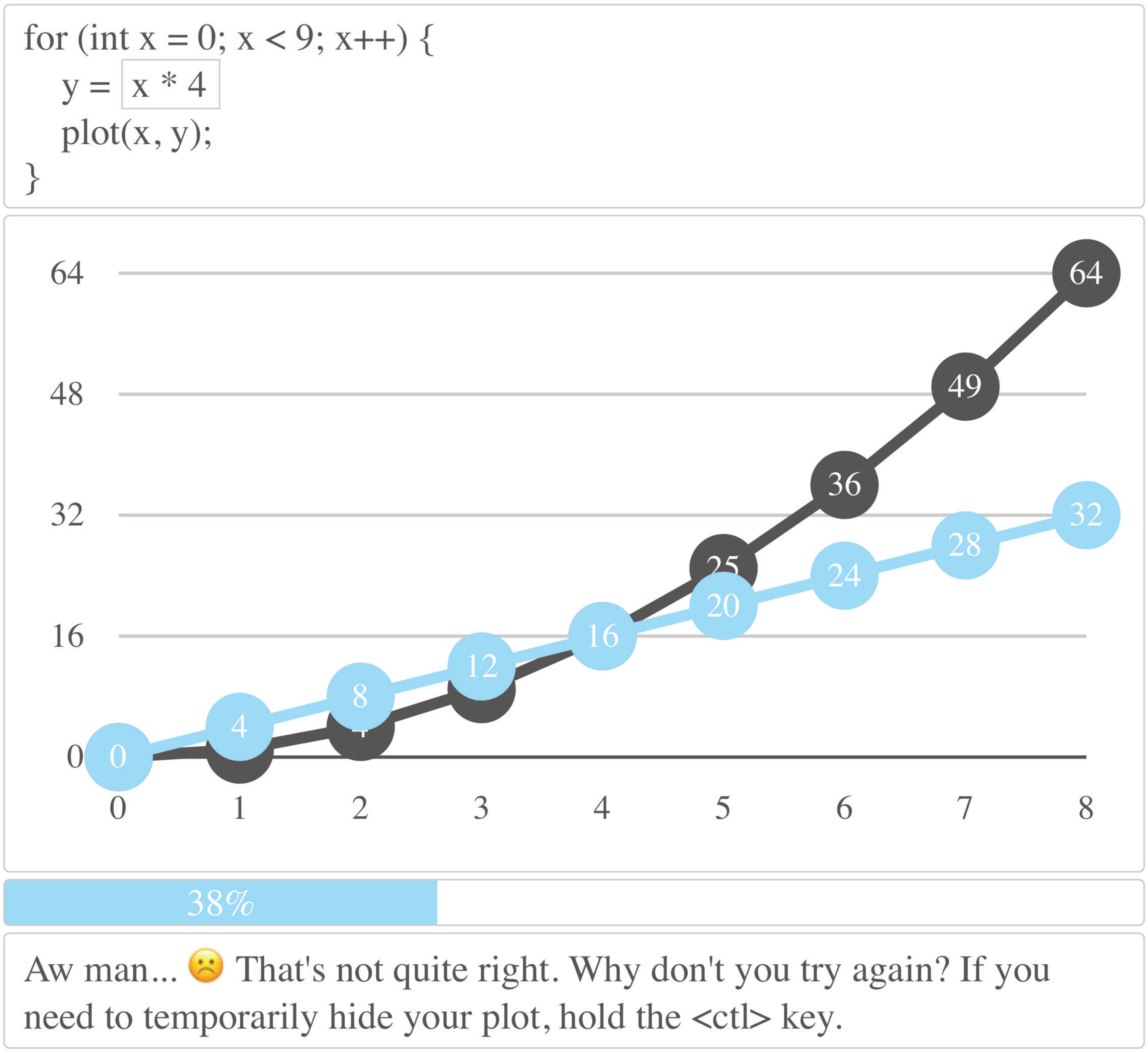
There are a plethora of existing educational programming games. In fact, an ITiCSE working group surveyed over 100 games for computer science education (Johnson, et al., 2016). None of the games surveyed, however, focus solely on for loops. For Play's limited scope reduces its interdependency with other programming topics and makes it easier for instructors to insert it into an existing curriculum.

For Play's design uses research results from the fields game design and computer science pedagogy to help players become proficient at writing code with for loops. The game helps students learn by scaffolding the coding process by providing partially complete code gradually increasing the difficulty level. It promotes active learning by using rapid, low-stakes, visual feedback of program execution. It engages and motivates players by using ephemeral, positive, visual feedback and personified error message reporting. And finally, it encourages players to create clean code with a reward system that values simpler solutions.

Game

For Play is an educational browser game written in JavaScript. In the game, players take the role of an eminent particle physicist overseeing a research lab. The lab's goal is to model the behavior of exotic particles in order to create a unifying theory of physics and ultimately find the answer to the ultimate question of life, the universe, and everything. The lab's technicians have gathered data on unexpected phenomena in the flux of dilithium crystals and have begun writing programs to simulate the phenomena. However, the technicians were unable to finish; they need the player's genius to finish the simulations' code.

The game's user interface presents the player with a small amount of program code, optionally in Python or Java/C++, that uses a for loop to repeatedly call a function to draw a line graph that represents the simulation output. However, the code is missing several key sections. The player must complete the code to solve the puzzle by filling in the missing pieces to reproduce the observed line graph and produce the correct simulation. Different levels of the game require different coding design patterns and omit different sections of the code to help the player practice different for loop programming paradigms. Figure 1 shows the user interface of For Play including the incomplete code at the top, the line graph visualization of the simulation in the middle, and the player feedback at the bottom. Note that the goal line graph is gray, and the line graph produced by the player's input is blue. A video demonstration of the game that better illustrates the gameplay can be found on its website (For Play, 2018).



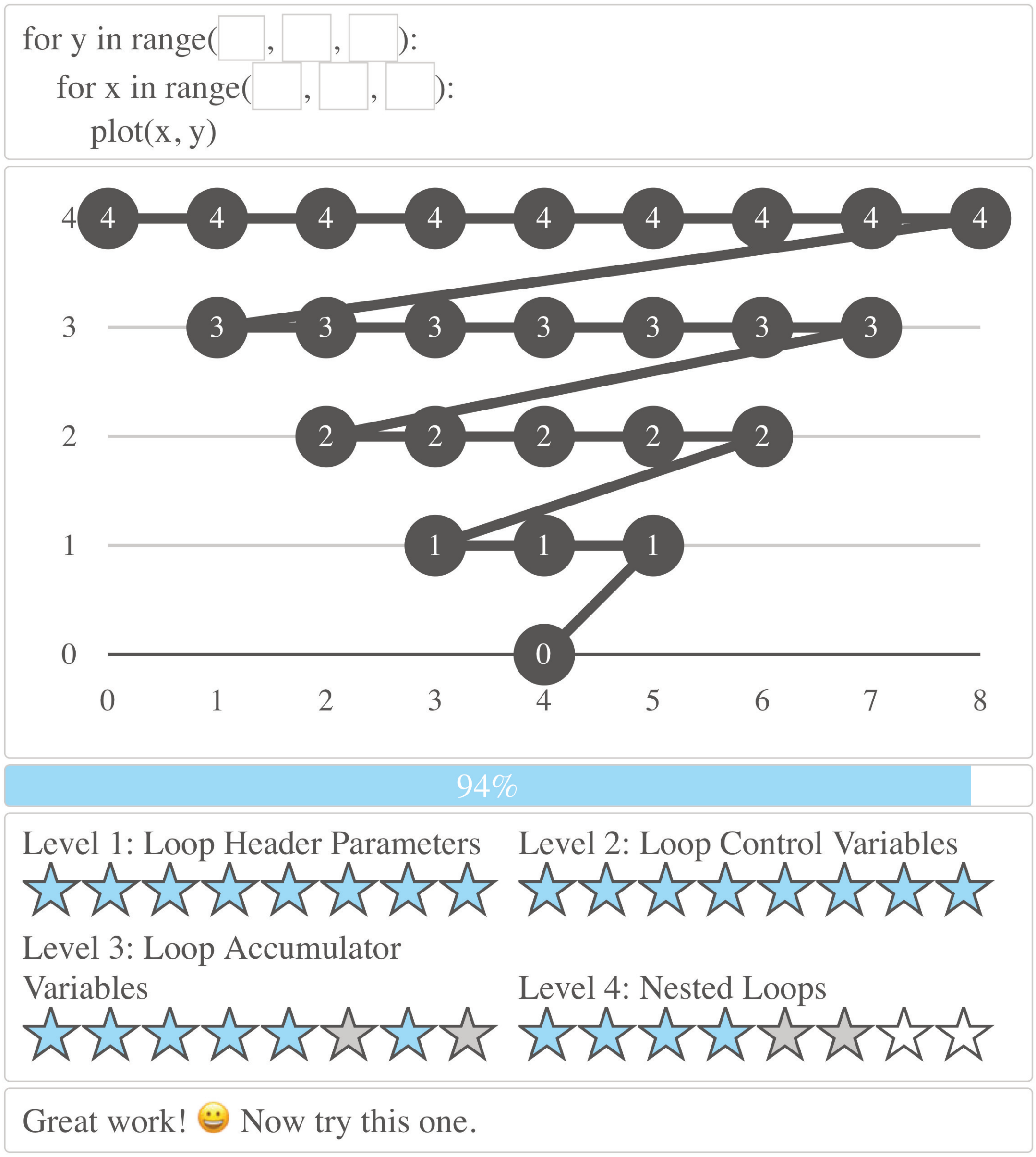
```
for (int x = 0; x < 9; x++) {
    y = x * 4
    plot(x, y);
}
```

Aw man... ☹ That's not quite right. Why don't you try again? If you need to temporarily hide your plot, hold the <ctl> key.

Figure 1. A Puzzle from Level 2: Loop Control Variables.

The game gives feedback to the player in the form of encouragement and program errors in the message pane at the bottom of the user interface. Above the message pane, the game displays the percentage of puzzles that have already been solved as a progress bar. Clicking on the progress bar reveals the puzzle selection pane. In this pane, the individual puzzles are represented as stars and are organized into different levels that focus on different for loop programming patterns. The player can select puzzles to attempt by clicking on the puzzle's star. The puzzle selection pane can be seen in Fig. 2.

The player can modify and reattempt a puzzle without any penalty. In this way, the game encourages the player to experiment and play with the code and to establish a connection between the visualization and the different editable parts of the code. Just like in programming, there are multiple solutions to every puzzle, but simpler solutions are better. Each puzzle is completed by reproducing the given graph, but players are encouraged to find the simplest solution by awarding colored stars for efficient solutions on the puzzle selection pane.



```
for y in range(   ,   ,   ):
    for x in range(   ,   ,   ):
        plot(x, y)
```

94%

Level 1: Loop Header Parameters

Level 2: Loop Control Variables

Level 3: Loop Accumulator Variables

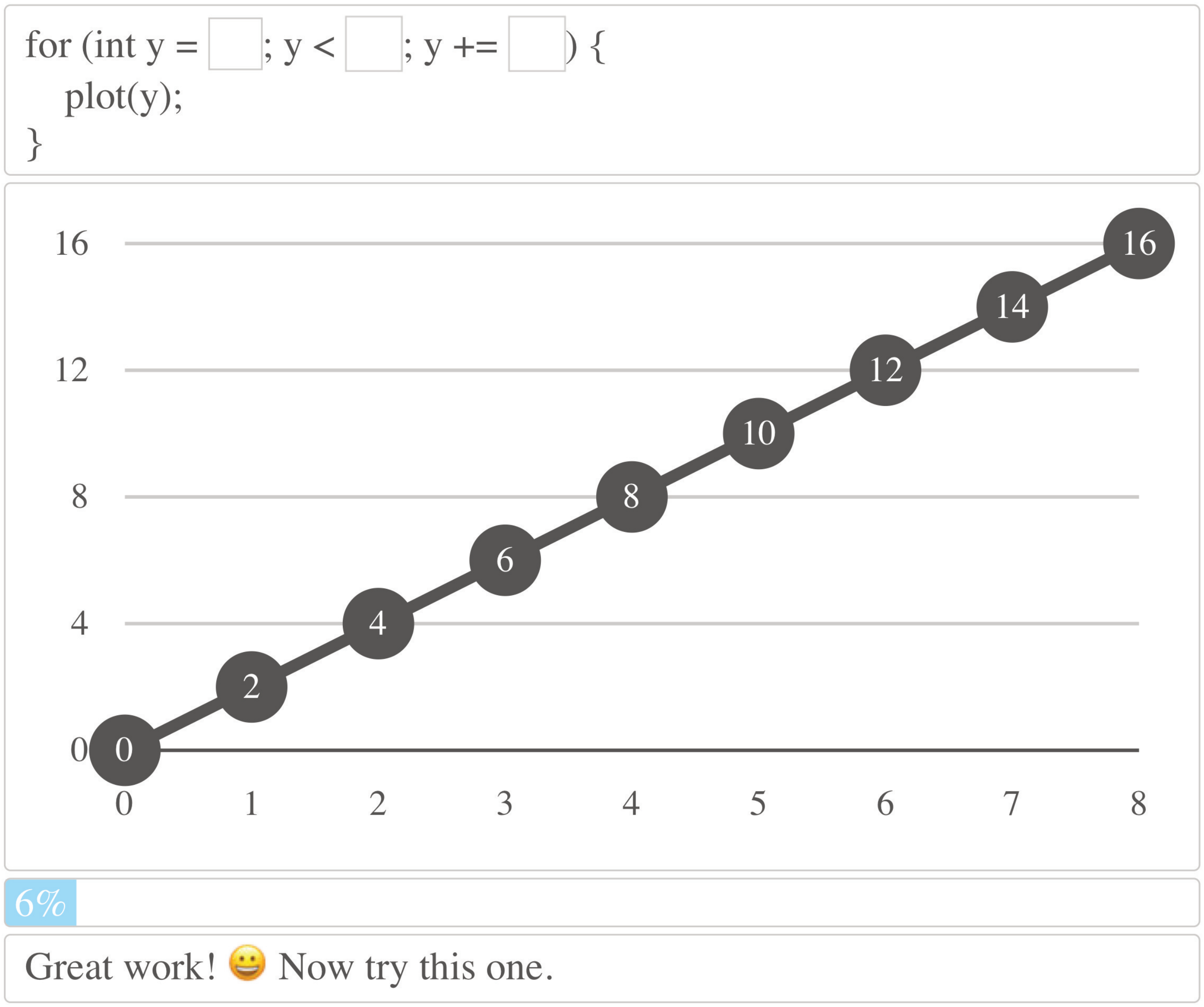
Level 4: Nested Loops

Great work! 😀 Now try this one.

Figure 2. A Puzzle from Level 4: Nested Loops.

Each of the levels of For Play focuses on different programming patterns that can be used to solve problems using for loops:

Level 1 – Loop Header Parameters: On level one, the player must complete the missing sections of a for loop header, the start, stop, and step in Python or the initialization, condition, and afterthought in Java/C++. The missing sections are the parameters of the for loop that determine the value of the loop control variable during each iteration of the loop. The loop control variable is used by the plot function in the loop body to produce the graph. See Fig. 3 for an example of a level-one puzzle. The player can experiment with the loop parameters' values and instantly see the impact on the graph visualization. This helps players learn how to write for loops by allowing the player to explore the relationship between the different parts of the loop and the behavior of the loop.



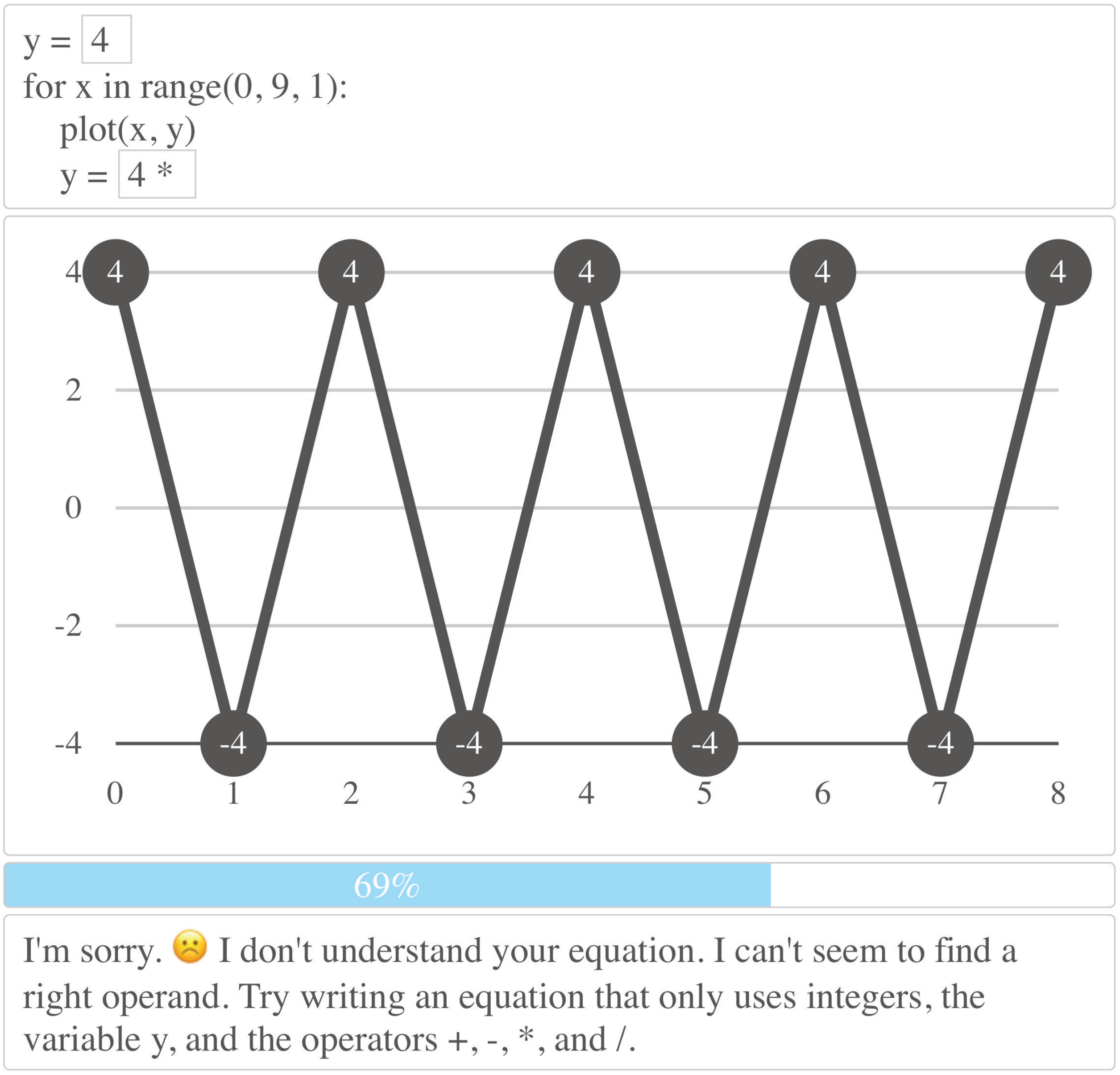
```
for (int y =    ; y <    ; y +=    ) {
   plot(y);
}
```

Great work! 😀 Now try this one.

Figure 3. A Puzzle from Level 1: Loop Header Parameters.

Level 2 – Loop Control Variables: On level two, the player cannot modify the loop header parameters. Instead, the player must create mathematical equations using the loop control variable to change the values passed to the plot function. See Fig. 1 for an example of a level-two puzzle. Some of the graphs in level two are identical graphs in level one, but because the player's control of the visualization is different, it encourages the player to make the connection between different approaches to solving the same problem. Level two also contains puzzles with non-linear plots that encourage the player to experiment with equations that are more complex and interesting.

Level 3 – Loop Accumulator Variables: On level three, the player cannot modify the loop header parameters or use the loop control variable. Instead, the player must use an accumulator variable. An accumulator variable is a variable that is defined before the for loop and is updated inside of the loop body using an equation that defines a new value for the accumulator variable using its current value. Figure 4 shows an example of a level-three puzzle and an accumulator variable. Using a variable's value in a calculation of itself is often confusing for novice programmers, especially when it is done inside of a loop. Puzzles on level three give players practice with simple uses of an accumulator variable by reusing graphs from levels one and two. Level three also allows players to establish the relationship between the accumulator variable's definition and its update statement.


```
y = 4
for x in range(0, 9, 1):
    plot(x, y)
    y = 4 *
```

I'm sorry. ☹ I don't understand your equation. I can't seem to find a right operand. Try writing an equation that only uses integers, the variable y, and the operators +, -, *, and /.

Figure 4. A Puzzle from Level 3: Loop Accumulator Variables.

Level 4 – Nested Loops: On level four, the player must complete the missing sections of two for loop headers, one nested in the body of the other. Figure 2 shows an example of a level-four puzzle. Nested loops can be tricky to write for novice programmers because the value of the control variable of the outer loop, which changes, can be used in the header of the inner loop. So, the interior loop not only runs repeatedly, but it can behave differently each time it runs. Puzzles on level four allow players to experiment with the header parameters of both the inner and outer loops which helps players establish the impact of the outer loop's control variable on the behavior of the inner loop.

Discussion

The design of For Play is guided by research in both game design and computer science pedagogy to make learning to code for loops easier and more enjoyable in several ways. Most importantly, the game's play mechanic, reading and writing code, is integral to the game's intended learning outcome, proficiency coding common loop programming patterns. Habgood and Ainsworth (2011) have shown that players learn more and play longer if the learning objective of an educational game is well integrated with and dependent on the game's primary play mechanic. Paras and Bizzocchi (2005) add that the gameplay should also include opportunities for reflection on the learning activity. For Play provides reflection opportunities by providing the player with rapid visual feedback input code execution and by not attaching game dynamics to the timing of the player's input. This gives the player unlimited time to reflect on their previous input which allows the player to draw conclusions and construct a new plan of action. This cycle is the process of experiential learning as described by Kolb (1984) which has been found to have learning benefits in many contexts.

The experiential learning feedback and reflection cycle of For Play also encourages play by using low stakes and rapid visual feedback. The game is low stakes because there are no penalties for incorrect solutions and there is no limit to the number of attempts or the amount of time between attempts. This encourages players to experiment with different inputs. For Play produces a visualization of code execution that allows the player to quickly see and comprehend why an input produces a particular output. Quickly seeing how a solution is incorrect helps players correct their mental model of program execution so that getting answers wrong is a useful part of the learning process. This process of experimentation is a form of active learning which has been shown to be beneficial to learning in many subjects, including computer science (Freeman et al., 2014). Odekirk-Hash and Zachary (2001) have also shown that rapid automated feedback can reduce the amount of help that learners need, and time spent solving problems without compromising learning outcomes.

Like all games, For Play uses rewards to influence player behavior. In the case of an educational game, the design and implementation of rewards are critically important as the rewards must not undermine the learning goals of the game. For Play uses several different rewards to encourage the player to complete the puzzles. The game uses both animations and textual messages for positive ephemeral feedback rewards. When a puzzle is completed, the visualization graph scales out while a congratulatory message is displayed. See the gameplay demonstration video on the game's website for an example of the animation (For Play, 2018). Bracken, Jeffres, and Neuendorf (2004) have shown that praise in the form of textual feedback can increase a user's intrinsic motivation and perceived ability.

For Play also uses rewards to encourage players to play longer and complete more puzzles with a completion progress bar, a puzzle completion checklist, and a level unlocking mechanism. In the puzzle selection pane, each of the programming patterns is represented as a separate level and each puzzle is represented as an outline of a star that becomes filled in when the puzzle is solved correctly. See Fig. 2 for an example of the completion checklist stars. Players can select any puzzle that has already been attempted in the current or previous level, but subsequent levels are not available until all puzzles in the current level are completed. Research by Mekler, Brühlmann, Opwis, and Tuch (2013) has shown that progress indicators and levels can increase play time without decreasing intrinsic motivation.

For Play also uses rewards for a secondary game objective. In computer programming, readable code is desirable because it facilitates communication on multi-person projects. So, the secondary objective of For Play is to create solutions that are simpler. On the puzzle select pane, the stars are filled in blue, instead of gray, when the player finds the simplest solution possible, as measured by the number of arithmetic operators used in the solution. Gaston and Cooper (2017) found that a similar three-star reward system encouraged replay and more thoughtful play, as measured by time spent searching for solutions.

Mayer's cognitive theory of multimedia learning posits that "people learn better from words and pictures than from words alone" (2014, p. 8). For Play uses a visualization to express both the puzzle objectives and the result of the execution of the player's input. For Play also animates the execution of a player's input so that the connection between the player's input code and the output visualization is made clearer. Vázquez-Iturbide, Hernán-Losada, and Paredes-Velasco (2017) found that using a visualization in a programming task increased learner motivation. While Grissom, McNally, and Naps (2003) found that students learn more from algorithm visualizations that are interactive. And the visualizations of For Play are inherently interactive.

For Play uses a visualization to provide feedback of program execution, but if there is an error or mistake in the player's input that prevents it from being correctly interpreted, a syntax error, For Play reports the error using personified messages. For example, in the Python programming language when a program contains an equation that is missing the right operand, attempting to execute the program will display the message "SyntaxError: invalid syntax". While For Play will display the message "I'm sorry. ☹ I don't understand your equation. I can't seem to find a right operand." See Fig. 4 for an example of this type of message in the game. Lee and Ko (2011) have demonstrated that personified feedback in a programming tool has a positive impact on learner engagement.

For Play is focused on learning to code for loops, so it uses a subset of the Python and Java/C++ languages. This greatly reduces the scope of issues that the player may encounter which also reduces the scope of errors that the game must detect. For example, if a loop does not run at all, it reports this as an error, while the Python and Java interpreters do not consider this an error as it is a useful behavior in some contexts. The reduced programming language syntax also makes it possible for the game to perform a lexical analysis of the player's input beyond whether the code is syntactically correct. This allows the game to detect common mistakes and to create customized error messages. Marceau, Fisler, and Krishnamurthi (2011) have found that error messages should use simple language and not make recommendations for solutions because learners tend to trust the error message's recommendation even if it is incorrect. So, For Play finds and succinctly reports errors, but it does not make any suggestions for how to correct the error.

Remembering the entirety of a programming language's syntax can be daunting for novice programmers. Moreover, making syntax errors can be aggravating because seemingly insignificant differences in code can cause fatal errors. For Play provides the player with much of the required syntax and allows the player to adjust just those parts of a code that are relevant to the programming pattern of a particular level. This is a form of scaffolding; a "process that enables a child or a novice to solve a problem, carry out a task, or achieve a goal which would be beyond his unassisted efforts" (Wood, Bruner, and Ross, 1976, p. 90). Structuring programming tasks in this way can give learners more confidence and a better understanding of the applicability of the subtasks being learned. Scaffolding has been shown to be useful in many educational contexts as it allows learners to solve complete problems before they have accrued all the skills that are required to solve complex and motivating problems. Linn (1995) has shown that scaffolding in computer science education can improve learning. It is worth noting that effective scaffolding requires fading, the reduction of the assistance given so that learners can gradually become more independent. For Play provides fading in the levels and puzzles that gradually scale up in difficulty. However, the game must also be part of a larger scaffolding that introduces the concepts of for loops before using For Play and challenges learners to independently write programs after playing the game.

Conclusion

I have described the educational browser game For Play that is designed to make learning to write code using for loops more fun. It does this by building off of research on rewards and feedback from game design research and on active learning and visualization from computer science pedagogy research. The game and its source code are freely available on its website (ForPlay, 2018). I encourage developers and educators to modify and use the game to make learning to code more fun for more people.

References

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: A retention study comparing graduating seniors with CS leavers. *ACM SIGCSE Bulletin*, 40(1), 402-406.

Bracken, C. C., Jeffres, L. W., & Neuendorf, K. A. (2004). Criticism or praise? The impact of verbal versus text-only computer feedback on social presence, intrinsic motivation, and recall. *Cyberpsychology & Behavior*, 7(3), 349-357.

Duckworth, A. (2016). *Grit: The power of passion and perseverance*. New York, NY: Scribner.

Ericsson, K. A., Krampe, R. T., & Tesch-Römer, C. (1993). The role of deliberate practice in the acquisition of expert performance. *Psychological review, 100*(3), 363-406.

For Play (2018). Retrieved from https://github.com/DurellBouchard/ForPlay

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23), 8410-8415.

Gaston, J., & Cooper, S. (2017). To three or not to three: Improving human computation game onboarding with a three-star system. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 5034-5039.

Grissom, S., McNally, M. F., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, 87-94.

Habgood, M. J., & Ainsworth, S. E. (2011). Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games. *The Journal of the Learning Sciences*, 20(2), 169-206.

Johnson, C., McGill, M., Bouchard, D., Bradshaw, M. K., Bucheli, V. A., Merkle, L. D., … & Zhang, M. (2016). Game development for computer science education. In *Proceedings of the 2016 ITiCSE Working Group Reports*, 23-44.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall.

Lee, M. J., & Ko, A. J. (2011). Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the Seventh International Workshop on Computing Education Research*, 109-116.

Linn, M. C. (1995). Designing computer learning environments for engineering and computer science: The scaffolded knowledge integration framework. *Journal of Science Education and technology, 4*(2), 103-126.

Marceau, G., Fisler, K., & Krishnamurthi, S. (2011). Mind your language: On novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 3-18.

Mayer, R. E. (Ed.). (2014). *The Cambridge handbook of multimedia learning*. Cambridge, United Kingdom: Cambridge University Press.

Mekler, E. D., Brühlmann, F., Opwis, K., & Tuch, A. N. (2013). Do points, levels and leaderboards harm intrinsic motivation?: An empirical analysis of common gamification elements. In *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, 66-73.

Odekirk-Hash, E., & Zachary, J. L. (2001). Automated feedback on programs means students need less help from teachers. *ACM SIGCSE Bulletin*, 33(1), 55-59.

Paras, B., & Bizzocchi, J. (2005). Game, motivation, and effective learning: An integrated model for educational game design. In *Proceedings of the 2005 DiGRA International Conference: Changing Views – Worlds in Play*.

Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of child psychology and psychiatry, 17*(2), 89-100.

Velázquez-Iturbide, J. Á., Hernán-Losada, I., & Paredes-Velasco, M. (2017). Evaluating the effect of program visualization on student motivation. *IEEE Transactions on Education*, 60(3), 238-245.