Reduct: A Puzzle Game for Children About Evaluating Code

Ian Arawjo
Cornell University
Ithaca, NY 14850, USA
iaa32@cornell.edu

David Li
Cornell University
Ithaca, NY 14850, USA
dml339@cornell.edu

Kevin Ma
Cornell University
Ithaca, NY 14850, USA
kym5@cornell.edu

CHI'17 Extended Abstracts, May 06-11, 2017, Denver, CO, USA
ACM 978-1-4503-4656-6/17/05.
http://dx.doi.org/10.1145/3027063.3048415

Abstract
We present *Reduct*, a puzzle game to teach programming language semantics to novices, especially children. Unlike previous puzzle games, *Reduct* gamifies the actual evaluation steps involved in executing code. Players discover behavior of language constructs through play by evaluating code snippets towards a goal. The game progression covers several basic concepts of JavaScript ES2015, including functions, Booleans, ternary conditionals, arrays, Array.map(), variables, and more. To help reduce self-handicapping behavior, code representations begin in a concrete form and fade to abstract notation over time.

Author Keywords
Educational games; learning games; programming games; novice programming

ACM Classification Keywords
K.3.2 [Computer and Information Science Education]: Computer Science Education; K.8.0 [Personal Computing]: Games

Background
Previous puzzle games for teaching programming focus on the process of program construction: the player writes a program, runs it, and rewrites it in an iterative process. In these "construction-first" approaches, players learn what

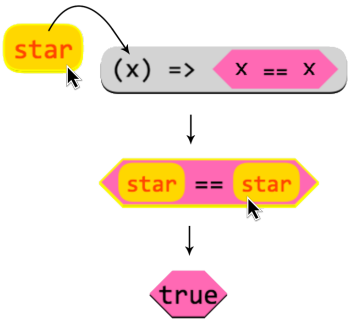

Figure 1: From top to bottom: A `star` constant is dropped over an anonymous function as input, binding `x` and producing `star == star`. Next, the player clicks to reduce the equality statement, producing the terminal value of `true`. Representations are in fully abstract form.

language constructs like 'rotate' mean – i.e., their semantics – through trial and error [2]. For instance, a player might be challenged to move a character toward a goal point by constructing a sequence of embodied actions like 'rotate 90, forward 2'. Since language semantics in constructionist approaches can only be inferred indirectly, it is difficult for players to discover the semantics of more advanced programming constructs through play alone [8]. Often these approaches also have the drawback of teaching a custom, rather than a general-purpose, programming language.

Core Mechanic

We considered how a game could teach language semantics to absolute novices *directly* through gameplay. We designed a puzzle game, *Reduct*, that gamifies the actual computation steps involved in evaluating code, i.e. reducing code to values upon execution. The player enacts these semantic rules with simple click and drag-n-drop actions. For *Reduct*, the set of rules follows the operational semantics [11] of a subset of the widely-used JavaScript ES2015 language. We hold that this approach could be applied to any evaluated language.

For example, Figure 1 depicts the evaluation steps involved in reducing a bound value. The player drops an expression over the input to a function to perform *substitution*, and then clicks the equality test to *reduce* it to the value `true`.

Gameplay Overview

Each level of *Reduct* has three areas of expressions: the board, the goal, and the toolbox (Figure 2). Using evaluation rules to enact a series of transformations, the goal in each level is to transform the set of expressions on the board to the set of expressions in the goal box. Expressions in the toolbox may also be brought into play to help reach the goal.

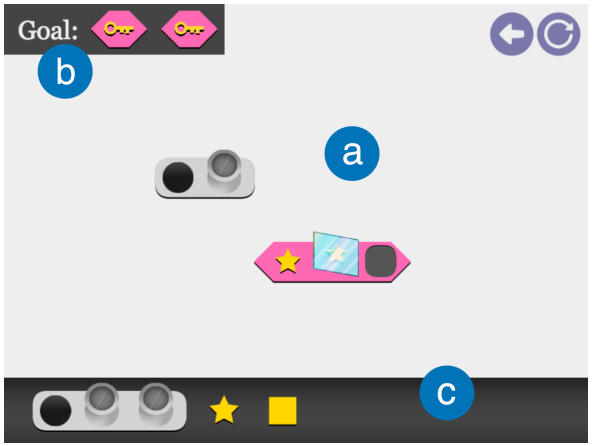

Figure 2: A level of the Boolean planet, with various expressions. Areas of play are (a) board, (b) goal, and (c) toolbox. The goal is to produce two `true` values, here represented in a concrete form as two keys.

In the context of Figure 2, the solution can be achieved by completing the equality test (in pink) with a second star, clicking to reduce it to a key (a `true` value), dragging the duplicator (leftmost in toolbox) onto the board, duplicating the key, and feeding one key through the Identity function that remains.

Concreteness fading

When learning mathematics, students may be initially averse to the appearance of abstract notation [5]. To help reduce potential self-handicapping, our design gradually "fades" the graphical form of expressions over time from concrete representations to abstract syntax (Figure 3). We were inspired by *DragonBox* (DB), a popular game for teaching algebra to children that adopts the method of concreteness fading [6]. DB begins with variables like `x` represented as a box

Figure 3: Green sparkles indicate a fade transition from concrete (left) to more abstract representations (right).

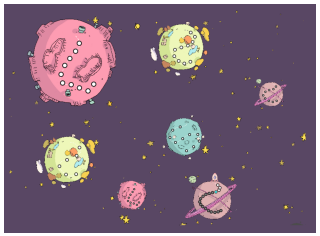

Figure 4: Concept select screen, showing planets on a starry background.

Figure 5: The title screen depicts Starchild floating in space amongst the stars.

and integers as insects, eventually fading to algebraic notation. In a large study, 96% of K-12 students who played DB for more than 90 minutes were able to solve three linear equations in a row with no errors in the game interface [7].

In *Reduct*, all expressions have at least one concrete and one abstract representation. The first level presents functions as having a hole for input and pipes for substituted output. This representation fades over several stages, eventually reaching the actual syntax of JavaScript arrow functions (Figure 3).

Progression

Our game progression adopts the theory of elaboration [9] by initially introducing a new concept in isolation, then mixing it with previous concepts over the course of a game section. Game sections are represented as planets in the menu interface (Figure 4). To encourage learning, we used theories of skill acquisition [3] to build competence in semantic rules by gradually increasing each level's difficulty (the complexity of the solution's search space) across each section. The released game progression covers several basic concepts, including functions, Booleans, equality tests, ternary conditionals, arrays, Array.map(), and variables. We are actively working on extending the progression to reach more advanced concepts, such as multi-line code sequences and `for` loops.

Narrative

We adopted a narrative to provide a fun framing to the game, while being careful to minimize its impact on the reward structure to avoid distracting from learning goals [4]. *Reduct*'s narrative follows the genderless Starchild, an alien searching for their home planet. Unfortunately, Starchild's spaceship keeps running out of rainbow dust. To enough rainbow dust to continue, players must complete every level on the planet they are marooned on.

Implementation

To ensure that the game was easily accessible to anyone with internet access and a basic computer, we decided to develop a web-based game with low graphics requirements and simple input methods. *Reduct* was implemented in JavaScript ES2015, the language it teaches, and tested on a Macbook running Google Chrome. A version of the game for this submission can be played at http://therottingcartridge.com/games/chi2017game/.

User Testing and Feedback

Our core progression and gameplay mechanics went through several iterations after many informal playtests with fellow students. We realized the meaning of the Goal box was not immediately apparent to some players. To fix this without adding text, we glowed pieces on the board in unison with those in the goal box upon victory. We also tuned the difficulty progression, making sure players didn't get stuck on beginner levels.

After our initial round of testing, an evaluation was conducted of an early prototype of *Reduct* both in-lab (24 participants) and online (over 300 players completed the game), verifying its core gameplay mechanic's ability to engage players, and measuring learning outcomes. Full results are presented as a full paper at this year's CHI [1]. Results from post-test questions asking players to type out expressions suggest that while players largely understood construct semantics, they often could not recall the precise notation used. A future version of *Reduct* might incentivize recall ability by requiring players to type syntax later on.

From the online deployment, players reported wanting to return to previous levels, which we fixed in this version with a menu select screen (Figure 6). We also received a few comments from parents whose children played the game,

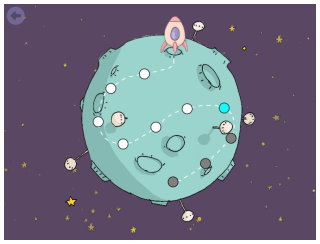

Figure 6: Level select screen for a section covering a single concept, represented as a planet.

suggesting that children found our design intuitive and engaging enough to finish a large portion of it, but found a later level requiring nested arrays too difficult to solve alone.

Future Work

In the future, we plan to evaluate whether *Reduct* can teach children at the K-12 level. After conducting an informal round of testing, we would deploy to a classroom as a limited workshop series. Of particular interest is whether the method of concreteness fading has an effect for a demographic likely to experience self-handicapping [5]. To test this, we would run a between-group study with concrete and fully-faded conditions, where learning outcomes in each condition would be evaluated both in-game (through a final sequence of levels), and on paper to test concept transfer outside the game environment. The paper test would be adapted from the language-independent FCS1 exam [10], which tests aptitude in CS1 programming knowledge.

Acknowledgements

We thank our advisors Erik Andersen, Francois Guimbretiere, and Andrew Myers for their close advisement of this project. Special thanks also to Cheng-Yao Wang, who helped conduct the evaluation, and to our early playtesters, Christine Geeng, Palashi Vaghela, Franziska Wittleder, Molly Feldman, Gabriel Culbertson, Dongwook Yoon, Huaishu Peng, Jeremy Fiume, and Luming Hao, whose helpful remarks and advice informed our design.

References

[1] Ian Arawjo, Cheng-Yao Wang, Andrew C. Myers, Erik Andersen, and Francois Guimbretiere. 2017. Teaching Programming with Gamified Semantics. In *CHI '17*. ACM, New York, NY, USA. (Forthcoming).

[2] Karen Ann Brennan. 2013. *Best of both worlds: Issues of structure and agency in computational creation, in and out of school*. Ph.D. Dissertation. MIT.

[3] Stuart E Dreyfus and Hubert L Dreyfus. 1980. *A five-stage model of the mental activities involved in directed skill acquisition*. Technical Report.

[4] Educationdive.com. 2016. Challenges persist when gamifying education. (December 2016).

[5] Emily R Fyfe, Nicole M McNeil, and Stephanie Borjas. 2015. Benefits of "concreteness fading" for children's mathematics understanding. *Learning and Instruction* 35 (2015), 104–120.

[6] Emily R Fyfe, Nicole M McNeil, Ji Y Son, and Robert L Goldstone. 2014. Concreteness fading in mathematics and science instruction: A systematic review. *Educational Psychology Review* 26, 1 (2014), 9–25.

[7] Yun-En Liu, Christy Ballweber, Eleanor O'rourke, Eric Butler, Phonraphee Thummaphan, and Zoran Popović. 2015. Large-Scale Educational Campaigns. *ACM TOCHI* 22, 2 (2015), 8.

[8] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. *Programming by choice: urban youth learning programming with scratch*. Vol. 40. ACM.

[9] C Reigeluth and R Stein. 1983. Elaboration theory. *Instructional-design theories and models: An overview of their current status* (1983), 335–381.

[10] Allison Elliott Tew and Mark Guzdial. 2011. The FCS1: a language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM CSE technical symposium*. ACM, 111–116.

[11] Glynn Winskel. 1993. *The formal semantics of programming languages: an introduction*. MIT press.