Resource Rush: Towards An Open-Ended Programming Game

Nicholas Lytle, Jennifer Echavarria, Joshua Sosa, Thomas W. Price
Department of Computer Science
North Carolina State University
Raleigh, NC United States
{nalytle,jechava,jsosa,twprice}@ncsu.edu

***Abstract*—Programming games provide players opportunities to practice and learn the fundamentals of coding in engaging ways. Many games have players program in block-based languages similar to environments like Scratch and Snap! as a means to scaffold student learning and reduce syntax errors. Block-based environments (BBEs) have been praised for their open-ended, constructionist designs allowing students to develop what they wish, express themselves, and explore the possibilities of the system. However, programming games tend to be more linear, usually designed as a fixed series of puzzles. We present Resource Rush, a game designed to resemble BBEs and present users with a game world that allows users to learn the fundamentals of programming in an open-ended game environment.**

I. Introduction/Related Work

There are currently dozens of programming games [6] [5] and many share key design features. One is the use of block-based programming languages as the means to program. Similar to block-based programming environments (BBEs) like Scratch [3] and Snap! [2], these block-based languages help scaffold students learning to program by easing the process of constructing code compared to text-based languages [16]. Many of these games fall into the puzzle genre where the game is set up as a series of increasingly difficult, self-contained levels that tasks players to solve a challenge by programming an agent. While the challenges sometimes can be completed in a branching order (ala Human Resource Machine), many have them in a fixed order. In Light-Bot [12], Program Your Robot [4] and BOTS [7], the challenge for each level involves navigating a robot through a maze through programmed commands. These are examples of games that Vahldick et al. cite as being "Logo-Like", taking cues from the famous constructionist learning environment, Logo [14]. Human Resource Machine [9] and 7 Billion Humans [10], both commercial programming games developed by the Tomorrow Corporation, tasks players to complete each level by having certain end conditions in the world met (e.g. variable be a certain value at the end of the run). For a completely novel setup, No Bug's Snack Bar, tasks users to program a chef agent that creates food and serves customers [13].

Block-based programming languages are common not just in these games, but also within BBEs. When block-based languages are used in BBEs, they are usually done to allow for creative, open-ended programming of multiple sprites or agents. Unlike BBEs, block-based programming games tend to be puzzle games controlling only one agent, are not open-ended and do not allow for creative, self-guided exploration. We feel there are extremely powerful affordances found in these BBEs that are missing in block-based games; we detail some of these below.

First, BBEs like Scratch and Snap! work by having a wide array of blocks (usually dozens) for users to use and experiment with, as opposed to the limited (usually less than a dozen) number of commands used in programming games. In addition, all of these blocks are available to the user from the start in a BBE, but not so in popular games where they must be unlocked through completing more levels. These blocks can be used in BBEs to control multiple agents (sprites) simultaneously, creating complex interactions. While some programming games like 7 Billion Humans allow users to program multiple agents, each agent must run the same developed script, unlike in Scratch or Snap!, where Sprites can have different scripts. In addition, the programming windows in BBEs are set up such that multiple scripts and behaviors can be developed for a single sprite. No Bugs Snack Bar is the only game described above that allows this as well. Further, in BBEs, custom blocks and procedures can be created and used to create more complex programs, but this is not found in most games, with only BOTS and Light-Bot allowing for the creation of a limited number of procedures.

Most importantly, BBEs like Scratch were designed to be constructionist learning environments centered around the idea that students learn by creating, and can guide themselves through the learning process by engaging in open-ended projects. This involves students being able to create their own computational artifacts (and share them with others around the world) given their own design ideas. In general, students in BBEs can engage in open-ended programming, creating challenges for themselves as well as having the freedom to tackle challenges in any manner they desire. We find this to be the most important feature of BBEs that are usually absent in programming games. By nature of the puzzle design, these games are not open-ended, instead explicitly giving tasks one after another for students to complete. Ensuring that a game still has this feature of BBEs was the main push for our game, Resource Rush.

II. Resource Rush: Current and Future Work

Resource Rush is a web game written in Blockly [8] designed to be an open-ended programming game for all ages. In the game, players take control of multiple agents in a farm-style setting similar to that of StarDew Valley [11]. Players can engage in multiple activities to build up their farm/land as they wish. While we only have farming and hunting mechanics developed as of now, we intend to expand the number of things players can mechanically do in the game (e.g. fish, cook, etc.) and build a game structure that allows for the easy extension of new activities. Each of these activities in the game have associated blocks related to completing these actions (e.g. 'harvest crop'). Like most BBEs, Resource Rush also has other blocks related to completing tasks in general (e.g., control structures) that can be accessed from the palette. Players can build programs to complete activities by snapping together blocks of code. Unlike most block-based programming games (but like BBEs like Scratch), players can have multiple scripts developed for an agent and individually select ones to run as well as save scripts to use later. We eventually will allow support for the development of custom functions that will allow players to complete increasingly complex tasks.

The game is designed to be open-ended in that students will not be presented a series of fixed levels. Instead, players will be given high-level goals or *missions* and will be tasked to complete these missions in the way they see fit. For example, if a student is tasked with 'Plant 100 seeds', they can choose to plant a harvest on a 20x1 plot 5 times or elect to make a giant 10x10 structure and only plant and harvest once. Other game missions like 'Collect 500 coins' will allow them to choose which of the activities that give money (farming, hunting animals etc.) they would rather build code for to complete. Like other games in the genre, completing missions will allow players to unlock additional features or cosmetic items, a common mechanic to keep people engaged in open-ended games. Though this is a game designed to teach programming, any of the agents the player is in control of can complete missions without programming using keyboard input. However, if players elect to program their agents using the block-based interface, the agents will enter Rush mode and complete their actions with a speed multiplier. This was done to encourage players to choose to program and illustrate how programming can speed up manual tasks.

Though we are still in the prototype stages, we believe that several new research directions open up thanks to design choices we have made creating an open-ended programming game. First, as our game does not force players to program at any point, we are able to investigate contexts in which players elect to code or do not, as well as research means of motivating code-averse players to willingly try. This might include structuring missions in such a way that it would be difficult to do so manually but easier programmatically, or give special motivating bonuses (like extra money or rewards) to those who complete missions through code.
Second, as ours is an open-ended game with no end, we

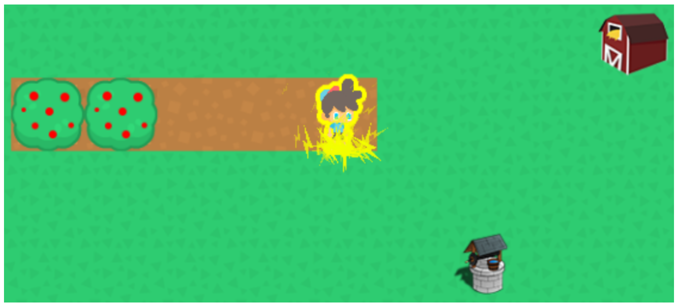
Fig. 1. A screenshot from Resource Rush. A player character (center) is planting crops (left) to be stored in the barn (top right). The player is doing this programmatically as the character is in 'Rush' mode (yellow sparkles).

will need a means of having enough missions for players to always be engaged with. This will most likely mean that we will have to procedurally generate missions, which has been used successfully as means of extending gameplay in Serious games [17]. Though there have been games so far that have procedurally generated content, our context opens up research directions in how to create appropriate missions for players in open-ended environments given their current play trajectory. This will include understanding how to generate appropriately skilled missions given a student's current level of programming prowess, potentially altering the design of the mission (i.e., providing starter code) to better scaffold novice players. In addition to generating the correct content, we will also have to investigate how to generate motivating educational contexts by creating missions that actually engage students in playing the game the way they want to play it while still learning to code.

While far off in development, we believe we can further extend this game by making it multiplayer. In-game, this might be represented as certain missions that players can work on together by visiting each others' worlds. This opens up tremendous opportunities in investigate social learning contexts in block-based environments. Further, though there has been recent work in systems like NetsBlox [18] that investigate collaborative block-based programming, this direction could unlock additional avenues of research in how students learn and collaborate in block-based environments as well as how students collaborate and learn in an educational game. Finally, though some BBEs like Snap! are open-source, most block-based games we know of are not. We are making a commitment now to ensure Resource Rush be open-source as we are hoping that this might encourage players who are engaged in learning to program through the game to actively practice their skills by developing for the game. This progression a player to developer might undergo is the very sort of agency we want to foster. Hopefully, by both playing and developing for Resource Rush, our attempt at an open-ended, ever-expanding programming game, we can truly show players the power of programming.

References

[1] Nicholas Lytle, Veronica Catete, Danielle Boulden, Yihuan Dong, Jennifer Houchins, Alexandra Milliken, Amy Isvik, Dolly Bounajim, Eric Wiebe, and Tiffany Barnes. “Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes”. In Proceedings of the 24th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 19) Aberdeen, United Kingdom.

[2] Brian Harvey, Dan Garcia, Josh Paley and Luke Segars. “Snap! :(build your own blocks).” Proceedings of the 43rd ACM technical symposium on Computer Science Education. ACM, 2012.

[3] Mitchel Resnick John Maloney Andrs Monroy-Hernndez Natalie Rusk Evelyn Eastmond Karen Brennan Amon Millner Eric Rosenbaum Jay Silver Brian Silverman Yasmin Kafai. “Scratch: Programming for all.” Commun. Acm 52.11 (2009): 60-67.

[4] Kazimoglu, Cagin, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. “A serious game for developing computational thinking and learning introductory computer programming.” Procedia-Social and Behavioral Sciences 47 (2012): 1991-1999.

[5] Michael Miljanovic, and Jeremy S. Bradbury. “A review of serious games for programming.” Joint International Conference on Serious Games. Springer, Cham, 2018.

[6] Adilson Vahldick, Antonio José Mendes, and Maria José Marcelino. “A review of games designed to improve introductory computer programming competencies.” 2014 IEEE frontiers in education conference (FIE) proceedings. IEEE, 2014.

[7] Rui Zhi, Nicholas Lytle, and Thomas W. Price. “Exploring Instructional Support Design in an Educational Game for K-12 Computing Education.” Proceedings of the 49th ACM Technical Symposium on Computer Science Education. ACM, 2018.

[8] Blockly — Google Developers. https://developers.google.com/blockly/.

[9] Tomorrow Corporation. Human Resource Machine https://tomorrowcorporation.com/humanresourcemachine

[10] 7 Billion Humans — Tomorrow Corporation. https://tomorrowcorporation.com/7billionhumans

[11] Stardew Valley — Eric Barone. https://www.stardewvalley.net/

[12] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. “Computational thinking in educational activities: an evaluation of the educational game Light-Bot.” Proceedings of the 18th ACM conference on Innovation and technology in computer science education. ACM, 2013.

[13] Adilson Vahldick, Antnio José Mendes, and Maria José Marcelino. “Dynamic Difficulty Adjustment through a Learning Analytics Model in a Casual Serious Game for Computer Programming Learning.” EAI Endorsed Trans. Serious Games 4.13 (2017).

[14] Mitchel Resnick, Stephen Ocko, and Seymour Papert. “LEGO, Logo, and design.” Children’s Environments Quarterly (1988): 14-18.

[15] Idit Harel and Seymour Papert. “Constructionism”. Ablex Publishing, 1991.

[16] Thomas Price and Tiffany Barnes. (2015, July). “Comparing textual and block interfaces in a novice programming environment”. In Proceedings of the eleventh annual international conference on international computing education research (pp. 91-99). ACM.

[17] Yetunde Folajimi, Britton Horn, Jacqueline Barnes, Amy Hoover, Gillian Smith, and Casper Harteveld. “A Cross-Cultural Evaluation of a Computer Science Teaching Game”. Proceedings of Games+ Learning+ Society. ETC Press, Pittsburgh, PA.

[18] Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L. Weeden-Wright, Chris Vanags, Joshua D. Swartz, and Melvin Lu. “A visual programming environment for learning distributed programming”. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 81-86). ACM.