Designing Visual Metaphors for an Educational Game for Parallel Programming

Santiago Ontañón
Drexel University
Philadelphia, PA, USA
so367@drexel.edu

Bruce Char
Drexel University
Philadelphia, PA, USA
charbw@drexel.edu

Jichen Zhu
Drexel University
Philadelphia, PA, USA
jichen@drexel.edu

Evan Freed
Drexel University
Philadelphia, PA, USA
efreed52@yahoo.com

Brian K. Smith
Drexel University
Philadelphia, PA, USA
bks59@drexel.edu

Anushay Furqan
Drexel University
Philadelphia, PA, USA
af627@drexel.edu

CHI'17 Extended Abstracts, May 06-11, 2017, Denver, CO, USA
ACM 978-1-4503-4656-6/17/05.
http://dx.doi.org/10.1145/3027063.3053253

Abstract

Modern computing is increasingly handled in a parallel fashion, however, little is known about how individuals learn parallel programming. This paper focuses on the design of an educational game called *Parallel*, designed for both teaching parallel programming education for CS undergraduate curricula, as well as for gathering insights into how students learn, and solve parallel programming problems. Specifically, we focus on the key challenge of choosing the appropriate *metaphors* in order to facilitate transference between parallel programming and the game. In this paper, we describe our design methodology and results from our preliminary user evaluations.

Author Keywords

Educational games; Parallel Programming; Visual Metaphors

ACM Classification Keywords

K.8 [General]: Games; D.1.3 [Concurrent Programming]: Parallel programming; K.3.2 [Computer and Information Science Education]: Computer science education

Introduction

Modern computing is increasingly handled in a parallel fashion. However, the conceptual shift form sequential to parallel programming is notoriously difficult to make. Different from the former, parallel programming requires systematic thinking skills in nondeterministic environments. Even accomplished programmers often encounter significant conceptual and practical challenges when writing parallel software. Although there is a growing body of best practices for when and how to teach parallel programming in Computer Science (CS) undergraduate curricula, less is known about how individuals come to learn the subject.

This paper describes our initial work toward the design of an educational game, *Parallel*, designed for teaching parallel programming in CS undergraduate curricula. As has been shown in the past, a key design decision in educational games is the choice of *metaphors* [6]. The first key design challenge in *Parallel* is the choice of *metaphors* used to visually represent the abstract programming concepts in the game: (1) *programming metaphor*: what is the metaphor used for representing "programming" (actually writing a program or something different), and (2) *visual metaphor*: do the game elements look like real-world objects, like trains, cars, etc., or like abstract symbols? We found that the metaphors used in existing sequential programming games did not translate well to parallel programming. We present the methodology we adopted in order to choose these metaphors (a combination of an iterative design process with user testing sessions). Our preliminary evaluations show that students were able to identify the key parallel programming concepts in the game, and that the design was successful in engaging them to play the game.

The remainder of this paper is organized as follows. We first summarize existing related work. After that, the design process we followed to design *Parallel* is described. Finally, we report on the results of a series preliminary user testing sessions used to refine our metaphors and to validate the game design. The paper closes with conclusions and directions of future work.

Related Work

There is a significant amount of literature on using computer games for teaching sequential programming, especially at the introductory level [4, 9]. Following the tradition of Logo, Scratch [10] and ALICE [7], new programming languages and often web-based environments such as code.org and the Khan Academy allow novice programmers to learn through creating their own game-like projects.

There is a large number of recent games designed to teach programming, or where programming is the basis of gameplay. In most of these games, the player is asked to solve specific coding tasks in a predefined game structure – how they progress in the game world is determined by the quality of their programs. Salient examples include, but are not limited to, *Cargo Bot*, *Check iO*, *Code Combat*, *CodeSpells*, *Gratuitous Space Battles*, *Light Bot*, *RoboCode*, *SpaceChem*, *HopScotch* and *Manufactoria* (the later being a significant inspiration to our final design). These games cut across different programming language abstractions (visual blocks, textual blocks and specific programming languages such as Java), different game genres (puzzle games, adventure games, and sandbox games), and different levels of programming competency (comprehension, writing, and debugging) [11]. Compared to sequential programming games, very few games exist that contain parallel programming concepts (such as concurrency, synchronization and non-determinism). Akimoto and Cheng [1] presented a high-level conceptual design of a Logo-like game, but to the best of our knowledge no actual game has been released. A recent textbook on Parallel Programming uses gaming concepts [2], but does not use any specific game.

Concerning metaphors, the idea that using visual metaphors to help learning programming has been discussed for decades [5], and it has been shown that a combination of textual and

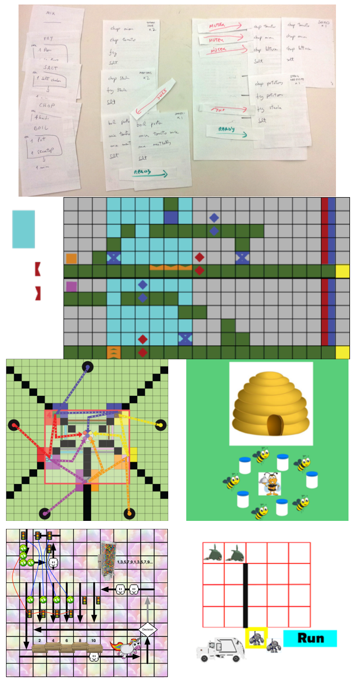

Figure 1: Sample screenshots and paper prototypes of early designs used for exploring the design space of parallel programming games.

visual metaphors seems to help learning [8]. In the context of educational games, metaphors have been found to play a profound role, by providing participants with a tool for provoking thoughts as well as being anchors for bringing the participant experience into the learning situation [6].

Moreover, we believe the methodology and insights about visual metaphor presented here are not specific to parallel programming, but also apply to sequential programming.

Game Design

Based on the curriculum of Drexel University's *CS 361: Concurrent Programming* class (taught by one of co-authors and project co-PI), we identified a three target conceptual areas the game was to focus on: (1) basic concepts such as *threads* and *speed-up* thanks to paralellism, (2) key differences between sequential and parallel programming (e.g., parallel programming is inherently non-deterministic, and thus the traditional concept of "debugging" is not very useful), and (3) synchronization (including concepts such as critical section, mutual exclusion, race conditions, signals, deadlock or starvation). The target audience is currently college students who are enrolled in an introductory parallel programming course. The methodology followed to design the game and metaphors consists of three main steps:

Step 1: design space exploration. The goal of this step was to generate a large set of initial game design ideas, so that we could start having a mental image of the whole space of possible designs. For that purpose, we tasked a group of several students to produce a prototype game design (one per student), then abandon it and come up with a new, different design. The only constraint they were given is that they should be able to represent levels or puzzles that capture typical parallel programming problems such as the *dining philosophers* or the *smokers problem* [3]. In this way, we obtained 12 different game designs with game mechanics ranging from puzzle games (6 of them), to simulation games (4), platformers (1) and sport games (1). We also explored a variety of game themes, with the designs covering themes such as: cooking, networking, robots, bee colonies, fantasy, abstract environments, basketball and traffic management. Figure 1 shows some screenshots, and paper designs of some of these ideas.

Step 2: design space analysis. The goal of this step was to compare and classify the set of 12 designs, and identify which are the main design *dimensions*, and their implications. After comparing all 12 games, we realized that the most defining factor was the level of abstraction of the *programming metaphor* employed to represent the parallel programming concepts in the game. Other dimensions that we identified include wether the games required skill or strategy or wether the role of the player is to create programs or to debug programs, but we quickly discarded them as not very relevant, since most games concentrated on one of their ends. Although the level of abstraction of the programming metaphor forms a continuous axis, three major categories were identified (from less to more abstraction):

- *Pure programming*: games where players need to actually program in a real programming language (however simplified), e.g., by directly typing. In these, the game visuals were used merely as an "execution visualization" tool.
- *Abstracted programming*: games where the concept of a "program" still exists, but in an abstract form (e.g., cooking recipes, or changing the visual configuration of robots). These abstract programs are then run on a visual environment (e.g., some game character follows a cooking recipe).
- *Environment as program*: in these games, there is no separate concept of "program"; the environment itself

defines the “program”, which players need to configure in order to define the the behavior of the game elements (e.g., placing traffic signals on a road). This was the most abstract metaphor for programming found in our game designs that was still able to represent the necessary parallel programming concepts.

After all the game design prototypes were placed along the abstraction spectrum, additional designs were produced to fill under-explored areas. Specifically, out of our initial 12 designs, only one fell close by to the *pure programming* side, and most laid somewhere in between the other two categories. So, some designs were produced to explore the *pure programming* side of the spectrum.

Once the axis was completely explored, we noticed that while *pure programming* games would provide the easiest *knowledge transfer* from learning how to play the game to learning parallel programming, since the mapping would be direct, the only benefit of the game would be the visual simulation. The other end of the spectrum resulted to be the most interesting. In particular, *environment as program* games allowed for more visual and intuitive level design than the other categories, which could potentially help students understanding the game faster.

Step 3: final game design. After full analysis of the design space, we chose to design a game in the less abstract end of the *environment as program* category. This allowed us to represent the programming concepts we needed, while maximizing the clarity of the levels by abstracting away from programming concepts not directly related to synchronization mechanisms in parallel programming.

The programming metaphor used in the final design is as follows. Programs are represented by *tracks* (i.e., roads or lanes), moving through them represents executing their associated programs. Certain key programming elements are present in the tracks (such as direction switches, etc.), and affect the behavior of the vehicles (which we call *arrows*, and represent “threads”) that move through these tracks. Programming thus amounts to placing elements along tracks. In a given level of the game, the player will see some specific track layout (Figure 2), and there will be a number of vehicles that will navigate these tracks. The goal of the player is to design a synchronization mechanism (by placing elements on the tracks) so that a collection of threads achieve a given task. Each game element (tracks, arrows, signals, semaphores) maps to a (parallel) programming concept, and it is designed so that intuitive inferences made by students based on them transfer to real programming. For example, since arrows following track represents threads executing programs, tracks are always only *one-way* (since programs cannot be executed backwards). The final game, called *Parallel*, is shown in Figure 2.

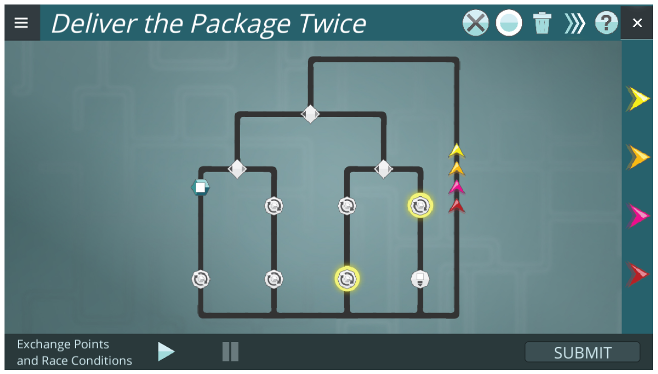
Deliver the Package Twice
Exchange Points and Race Conditions
SUBMIT

Figure 2: A screenshot of *Parallel*, in a level with four *arrows* (four threads), introducing the concepts of *race condition* and of *message passing* (represented by what we call *exchange points*).

Thanks to this programming metaphor *Parallel* trivially satisfies first and third of the requirements above. To capture the second requirement, we introduced two concepts (inspired by the *Manufactoria* game): (1) that of *running* a level (arrows will move at random velocities) and (2) that of *submitting* the solution for grading (assuming that the game is being played as part of a class). Running the level can be used to get intuitions of what the current solution will do, but offers no guarantees. When a player *submits* the solution, a model checker performs exhaustive search, verifies that the solution will always work, and provides a visual rendering of a failing scenario if any exists. The idea is to learn that debugging parallel programs is hard because the fact that a parallel program happened to run correctly one or ten times, does not imply that the program will always work.

The final design decision concerns the choice of *visual metaphor*, i.e. wether the tracks look like roads, train tracks or something else, and whether the other game elements are also depicted by some physical counterpart (trains, cars, etc.) or are abstract symbols. In order to make that decision, we employed a series of user tests aimed at discovering the types of inferences that students were making given the game visuals.We describe these below.

User Testing

We have performed three preliminary user tests during the design of *Parallel*, with the goal of choosing a visual metaphor, and evaluating our programming metaphor. This section briefly summarizes these tests.

Round 1 Testing. *Goal*: assess the validity of the *programming metaphor* and *visual metaphor*. *Context*: we showed a very early prototype to 2 of the top students who recently passed the concurrent and parallel programming class at Drexel University. This prototype used a “cars driving on roads” visual metaphor. There was no help screens in the game and a fact sheet was provided in a piece of papers. *Results*: in general the users liked the game idea, and said things like “*good for new programmers since they usually use print statements to see what’s happening*”. However, several comments pointed out that our visual metaphor was not working. The problem is that students are very familiar with the concepts of cars and roads and the physical interactions that occur when, say, two cars meet in the road (they need to dodge each other), which did not have an analogous in parallel programming. Also, they were assuming that roads would have “lanes”, but given that “roads” represented lines of code, the concept of “lane” didn’t translate to parallel programming well either. As a result of this study, and after exploring a number of other visual metaphors (like trains) we decided to move away from those and use an abstract visual metaphor where threads are just represented by “arrows” moving through “tracks”.

Round 2 Testing. *Goal*: evaluate the new visual metaphor, and understand what students were learning. *Context*: we showed a version with the new visual metapor to a group of 5 CS graduate students. No fact sheet was given to them this time, only the on-screen instructions of the game itself. *Results*: unfortunately results from this study were not very significant due to a number of UI issues (16 our of the 37 written comments we logged refer to UI problems) and problems accessing the help screens (8 out of 37 comments). However, 100% of the students reported seeing the analogy between parallel programming and the game, and made comments such as “*It will be nice to have a tool like this in a IDE for multi-thread debug*”. Concerning their understanding of parallel programming concepts thanks to the game, we asked a few preliminary questions concerning certain parallel programming concepts after playing two game levels, and only 50% (5 out of 10) of the answers

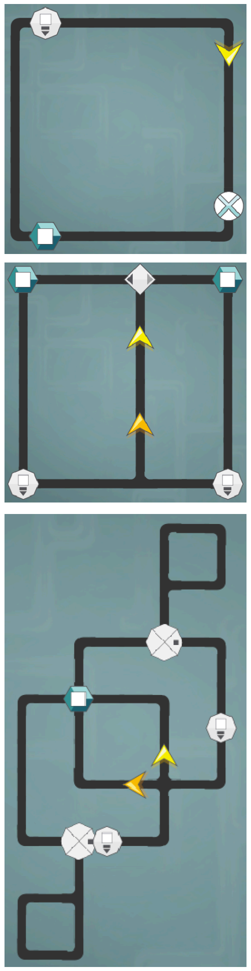
Figure 3: Layout of the first three levels used on our round 3 user testing.

were correct. However, these numbers are both affected by the UI/help screen issues and by being a small sample.

Round 3 Testing. *Goal*: re-run of round 2 testing with the UI/help issues fixed. *Context*: a refined prototype with all the UI and help screen issues fixed to 2 CS students who had previously taken the concurrent and parallel programming class. The first 3 levels used for this evaluation are shown in Figure 3. *Results*: again all students saw a connection between parallel programming and the game, and both of them were able to correctly identify which (out of: critical section, starvation, race condition, deadlock, message passing) were present in two of the levels of the game. In their feedback they mentioned things like "*Yeah. The race conditions were really apparent that was really good example of race conditions*". However, only one of them was able to identify one of the largest levels in our game as an instance of the *dining philosophers* problem (both students expressed familiarity with the problem). Both students enjoyed playing the game (comments include "*I didn't know what I was getting into and I, and I like it*"), but one of them mentioned that he had difficulties thinking in terms of parallel programming to solve some of the larger levels ("*I was able to get it done one way according to the objective but I mean I didn't really know what was really going on I just kind of fiddled around until maybe it worked so it's kind of hard to grasp what is supposed to go on*").

In summary: (1) the abstract visual metaphor worked better than the physical metaphor, since it did not misguide students in making the wrong inferences; (2) level design needs to be improved to encourage parallel programming thinking and discourage trial and error (we are currently considering improving UI hints, and better track highlighting which can help students look ahead into the implications of their solution without needing to constantly run the level).

Conclusions

This paper has presented the design process and initial evaluation of *Parallel*, an educational game designed to teach parallel programming concepts to undergraduate students. We identified the choice of programming and visual metaphors as the key design challenges, and presented our methodology to determine those. The main contributions of this paper are a methodology for choosing metaphors, and insights into the type of visual metaphors that are more effective for parallel programming specifically. The preliminary results of our user evaluations indicate that our programming metaphor is solid, as most students are able to see the connection between the parallel programming concepts we are trying to teach, and the corresponding game concepts. Moreover, our evaluation has also showed that an abstract visual metaphor better supported the learning processes of students than physical visual metaphors (common in existing sequential programming games) by not misguiding them in the inferences they made about the game object behaviors. As part of our future work, we plan to design a level progression for the game that serves as a companion for Drexel University's concurrent and parallel programming classes, and use it to run a large study on knowledge transfer between the game and parallel programming. This study will be the basis for our future work on player modeling and content generation, in order to achieve personalized parallel programming educational experiences.

Acknowledgments: This project is partially supported by Cyberlearning NSF grant 1523116.

Additional Authors: Michael Howard (Drexel University, email: mxh23@drexel.edu), Anna Nguyen (Drexel University, email: adn33@drexel.edu), Justin Patterson (Drexel University, email: justin.h.patterson@gmail.com), and Josep Valls-Vargas (Drexel University, email: jv384@drexel.edu).

References

[1] Naoki Akimoto and J de Cheng. 2003. An educational game for teaching and learning concurrency. In *Proceedings of the 1st International Conference on Knowledge Economy and Development of Science and Technology (KEST'03), Honjo, Japan*. 34–39.

[2] Ashish Amresh and Ryan Anderson. 2014. *Parallel Programming Using Games: A Hands-On Approach*. AK Peters, Ltd.

[3] Allen Downey. 2009. *The little book of semaphores*. CreateSpace Independent Publishing Platform.

[4] Casper Harteveld, Gillian Smith, Gail Carmichael, Elisabeth Gee, and Carolee Stewart-Gardiner. 2014. A design-focused analysis of games teaching computer science. *Proceedings of Games+ Learning+ Society* 10 (2014).

[5] Helen Duerr Hays. 1988. Interactive graphics: A tool for beginning programming students in discovering solutions to novel problems. *ACM SIGCSE Bulletin* 20, 1 (1988), 137–141.

[6] Thomas Duus Henriksen. 2014. What Role do Metaphors Play in Game-Based Learning Processes? In *Advances in Game Design and Development Research*. Nova Science Publishers, Incorporated.

[7] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1455–1464.

[8] Elspeth Mckay. 1999. Exploring the effect of graphical metaphors on the performance of learning computer programming concepts in adult learners: a pilot study. *Educational Psychology* 19, 4 (1999), 471–487.

[9] Michele Pirovano and Pier Luca Lanzi. 2014. Fuzzy Tactics: A scripting game that leverages fuzzy logic as an engaging game mechanic. *Expert Systems with Applications* 41, 13 (2014), 6029–6038.

[10] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[11] Adilson Vahldick, António José Mendes, and Maria José Marcelino. 2014. A review of games designed to improve introductory computer programming competencies. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 1–7.