Disappearing Happy Little Sheep

Changing the Culture of Computing Education by Infusing the Cultures of Games and Fine Arts

Adrienne Decker
Andrew Phelps
Christopher A. Egert

This article explores the critical need to articulate computing as a creative discipline and the potential for gender and ethnic diversity that such efforts enable. By embracing a culture shift within the discipline and using games as a medium of discourse, we can engage students and faculty in a broader definition of computing. The transformative quality of games can be explored at many levels: the creative person using technology as their palette to explore new modes of expression, the process of technology creation forming a unique and intimate relation between the user and the content, and the gateway of emphasizing the creative process to form a deeper connection between the creative personality and the consumer of such technologies.

Adrienne Decker is Assistant Professor in the School of Interactive Games and Media at the Rochester Institute of Technology (RIT). Her research focuses on media-centric computing education with an emphasis on diversity and outreach. She is the current ACM SIGCSE treasurer and a member of the AP Computer Science A development committee (e-mail: adrienne.decker@rit.edu). **Andrew Phelps** is the founding director of the Center for Media, Arts, Games, Interaction and Creativity (MAGIC) at the Rochester Institute of Technology (RIT). He is also Professor and founder of the RIT School of Interactive Games & Media, and a faculty fellow of the RIT School of Individualized Study (e-mail: amp5315@rit.edu). **Christopher A. Egert** is Associate Director of the Center for Media, Arts, Games, Interaction and Creativity (MAGIC) at the Rochester Institute of Technology (RIT). He is also Associate Professor in the RIT School of Interactive Games and Media in which he teaches Game Engine Design and Development (e-mail: caeics@rit.edu).

Introduction

When students begin their journey into the world of creating video games, they soon discover that they must balance their passion and enthusiasm for creating interactive experiences with a complex fusion of skills from technology, science, and art. There is a tension between the technical complexities required to construct a game experience and the need for story and entertainment. It is easy to recognize the juxtaposition of the formalism versus the need for a vibrant and creative playground to express concepts and ideas. For students entering the craft, their first exposure to realizing their vision is the art of programming. In higher education, there exists a continued divide between a classic view of the field as a mathematical science, and a hybridized, applied form that seeks to engage general undergraduates in *algorithmic and computational thinking*.

We propose a different approach. We seek to directly and concretely think of computing education as an artistic practice. In other words, the act of programming within itself is a form of creative expression. We believe that this creative expression can be achieved through the design and development of video games, which are, in themselves, a uniquely creative and computational medium. Although many people extoll the virtues of using games for learning, we are not thinking of playing games as a means for teaching computer programming or other computing concepts, but rather as a domain for formalizing and contextualizing the learning of those concepts. Games allow for explorations in creativity like few other types of programming projects can provide. There is no agreed upon single right answer to what constitutes a compelling and engaging game experience. The creators of games are allowed to express themselves through the narrative of the games or through the game play itself. In doing so, the practice of technology skills becomes a learned proficiency, allowing us to create ever more complex works of expression. Games are often deceptively simple in their most basic form, but can take a lifetime of iteration as they grow more complex. Like art, they are never truly finished or perfect. The work inspires the artist to forever invest in learning more.

But exploring games as an art of expression goes much deeper and illuminates parallels to processes utilized when studying the fine arts such as drawing, illustration, and painting. Recognizing these synergies and shifting the educational paradigm to embrace some of these ideas from the fine arts can transform the culture of the computing education classroom in new and unique ways. We present the idea of a drawing pad from a typical art class. In such a class, students are told to fill it with drawings as part of the educational process. We would really like our students to fill up their drawing pads with code. When we focus on games, we want our students to fill up their drawing pad with code that results in the realization of game design ideas. Ultimately, we want students to think computationally. Unfortunately, in conveying our discipline to students, we can create two significant hurdles.

The first hurdle is what we ask students to do. We tell learners to create very specific programs. We measure success through the realization of code that generates the correct output, that passes the test case(s), and that conforms to the specification. We ask students to build things that reinforce our own ideas about what constitutes the field, about what it means to be a computer scientist, and about what is important. The second hurdle is the barriers the current technologies put in place. At the beginning of their learning, students will spend an inordinate amount of time learning the complexities of a language, compiler, IDE, debugger, and other tools of the developer. For some, these two hurdles combine to create an impenetrable wall creating a culture of exclusion based on skills and technological prowess instead of fostering a culture of inclusiveness and creativity. We further illustrate these hurdles with two stories about passion and creativity.

'Nice Sheep'

All children draw, nearly without exception. They draw with crayons, marker, pencil, paper, and chalk. They express their ideas on paper, on sidewalks, and on walls. Most young children are proud of their drawings, are engaged in their creation, and most importantly ***believe themselves capable of drawing****. The thought that they "can't draw" is not a part of the early experience, and this is reinforced by parents, friends, work-on-display on the family refrigerator, and an abundance of positive feedback with little to no critique. At some point, for some children, that all changes. One day, they are drawing the picture of a perfect day and starting with the most beautiful sky full of puffy white clouds. Along comes a well-meaning adult, excited to praise the child for their wonderful work. The adult says, "nice sheep."*

In an instant, the child is face to face with the failure of the work, and begins to internalize that he or she "can't draw," that the drawing is "not good," or does not conform to the conventions of a particular form, style, or interpretation. All too often in computing education, we create a system of haves and have nots. Students internalize their inabilities as failure, causing both students and faculty alike to believe in something like a "geek gene" that ultimately controls whether or not they will succeed. This tearing down of self-confidence leads to a lack of self-efficacy, which has been shown to be a key factor in success in the introductory classroom (Wilson, 2002), and this divisive barrier serves to chase newcomers away (Furger, 1998; Scragg & Smith, 1998)

By channeling creative output through games, we shift the focus to creative exploration instead of pursuit of a singular correct answer. We can show that there is value in the process of creation and that defeats are temporary. The process of realizing the vision of creation and the potential for misinterpretation is merely a step in the process to achieving the finished product. Instead of focusing on the correctness or efficiency of a particular solution, the focus is put on the experience the game provides. From a cultural perspective, the instructor in this type of computing classroom creates opportunities for students instead of limiting them. The instructors need to recognize and encourage alternative solutions, not automatically dismissing them as incorrect. In this model, assignments are not a process by which the instructor filters the technological prowess of the student, but rather helps the student to iterate and expand upon their ideas to achieve the ultimate goal. Finally, as instructors, we must remember to remove our biases and preconceptions and ask the students if they made clouds or sheep.

Disappearing Ink

A young student has signed up for an art class. Excited, ready, and willing to do the work, the first assignment is to complete a drawing on a special paper with a pen that the instructor has handed out. The drawing is coming along beautifully and the student is satisfied that they have met the criteria for the assignment and created a drawing they are proud of. At the start of the next class, the instructor asks each student to turn in their drawing. Upon handing it to the instructor, the paper activates the magic properties of the ink from the pen and it disappears. The student has nothing. It is all gone.

Meet the compiler. In fact, it's not just the compiler, it is a gauntlet of technological trials that students must overcome to even experience the bare minimum of their programming efforts. From the start, students must overcome learning the syntactic vagaries of a particular programming language. From there, students must master the obscure process of compiling code while attempting to master a complex IDE or an extremely esoteric command line process. At this point, if the code does not meet the compiler's exacting expectations, students must decipher the cryptic error messages it produces. All of that even before they can attempt the process of running the compiled code and then starting to debug it if there are errors. This process also mimics how we convey specification to students, often demanding exact conformity, and only valuing those solutions that meet the rigid standards or the instructor's vision.

As a matter of practicality, we must recognize that some of these barriers are currently an essential part of the process when creating any type of game or program. There is a recognition, however, that these are in fact barriers to student success (Garner, Haden, & Robins, 2005; Rodrigo & Baker, 2009). What is unfortunate is that students spend an inordinate amount of time and focus on this level of minutia instead of looking at the holistic and formative process of creating their solution. The instructors often convey the idea that success comes from satisfying the specification rather than playful exploration and discovery of alternative solutions. When students are given the task to build a game, instructors will often give a specific requirement about the game (e.g., it must use a linked list), but the exact nature of the game is left up to the student. They are not required to implement the exact vision of the instructor and would earn more praise for innovation and exploration of a creative approach even if the end product lacks completeness and polish. This changes the nature of success in the classroom from one of conformance to an inflexible specification towards ingenuity and imagination.

Tools such as block languages, diagramming toolkits, and functional prototyping tools have all been developed with the goal of scaffolding students through technological barriers. Environments such as *Scratch* (Resnick *et al.*, 2009) and *Alice* (Cooper, Dann, & Pausch, 2000) make visual programming easy and provide a sense of user feedback in that the direct manipulation of the block elements always results in a tangible output from the system. Both of these environments are playful and provide an appropriately scaffolded sandbox to encourage students to express themselves through games and other interactive media experiences. As instructors, we must remember to pull the focus away from technology specific details and instead provide an environment that ensures the student's ink doesn't disappear.

Want to Get to Carnegie Hall? Practice!

One of the key educational tenets of the creative arts is repetition. Students in creative disciplines such as art, music, and dance are told to practice every day, as much as possible. They are told that the more they practice, the better they will be. Educators in these fields recognize, however, that practice alone is not enough. In order for the practice to further one's skill, the student first must find their passion for the craft. This allows the students to simultaneously explore their passions and improve their skills. For example, we often hand an artist a sketchbook and tell them to "fill it up." We ask students to draw what they see and express what they feel. We ask students to find their voice and to convey their message. During the process of filling the sketchbook, we might, like Bob Ross, encourage them to continue to paint "happy little trees" over and over again until they can do it reliably (but still recognizing the variations and 'happy accidents' that he praises in his popular televised approach).

Once the book is filled, we evaluate the work by first asking students to reflect upon their creations and their journeys. The instructors would provide guidance and assist in the reflective process while igniting the spark for the first drawings in the next sketchbook. The educational value is in the practice, *as* the outcome. When the focus is game development, we can encourage the spark of creativity that exists within students who proclaim this passion. By creating a culture of practice in the computing classroom, we can contribute to the reinforcement of skills through repetition. Further, by including reflection as an integral part of the process, we spur each student's growth. By entwining passion, practice, and reinforcement, we transform the learning process from a chore to one of the desire to improve one's craft.

Towards a Daily Ritual of Computational Practice: Programming Studio

What if we created a course that said simply: program a game. Program a game every day, and show, discuss, and critique the results, *a la* a studio model. We highlight the expressive and engaging nature of the medium as a form of art. We allow for the creation and exploration of artifacts and forms that are of interest to the students and sparked by their personal passions. With games, we have the ability to motivate by enabling the authorship of interactive stories and experiences, such as when we ask the students to learn to program in platforms like *Alice* (Cooper, Dann, & Pausch, 2000) and *Greenfoot* (Kölling, 2010) or to extend and modify already existing game platforms like *Minecraft* (Zorn, Wingrave, Charbonneau, & LaViola, 2013).

Much like art, in a programming studio, we would like them to mimic. We want them to copy the masters, the structures, and the patterns. Copying in this case is, in fact, totally appropriate and should be encouraged. However, the mimicry must be balanced with relevance, creating connections to the culture and technology students are exposed to every day, learning from our past to create our future. We can use the case of copying the masters as a way to recognize the history of the discipline, the lessons learned from the past, thereby planting the seed upon which to build the next great idea.

How can we capitalize on, and embrace the motivation and message of each and every young person as a potential artist of technology? To best navigate the waters of inclusiveness, we should provide a platform for the students to express themselves as individuals, as well as in teams of collaborators and a wider culture of creativity. The classroom should put on display what they find exciting and interesting. By giving students a creative outlet through games, we provide each person a unique voice and a unique vehicle for expressing that voice. Whether the outcome is pure entertainment, games for learning, games for behavioral reflection, or games as a message platform, the ability to express the richness and diversity of the creators is limitless.

Critique, Assessment, and Verification

A hurdle to adoption of the programming studio is shifting the current assessment culture in the computing classroom. Current assessment practices, such as automated grading frameworks (de Souza, Maldonado, & Barbosa, 2011; Williams, Bilalac, & Liu, 2006), mentoring frameworks (Daly, 1999), unit testing approaches (Marrero & Settle, 2005), and systems designed to capture and disseminate feedback from mentor to learner (Kumar, 2005), often focus upon the functional requirements of the assignment and will do little to address outcomes beyond the technological and algorithmic goals of the required task.

In contrast, creative works typically undergo a process of critique, reflective of the unique nature of each piece. Groups of artists analyze the final product as well as the work that went into them, the process by which they were made, and the message they convey. For beginners, programs are seen to simply materialize at the moment of execution. The new practitioner is often left to wonder by what sorcery program correctness actually occurs, and the educator is left guessing as to the process of revision that a student used to arrive at their final program.

Looking deeper at the notion of critique, we are not simply looking for functionality meeting specifications or correctness, but rather looking at the whole. Further, critique is of the work, not of the individual. When we ask if someone's code "meets the spec" or "passes all the test cases" we often prescribe the success or failure to the individual. When we critique a game, we are looking at the work as the success or failure, not the individual. We do not use the critique of a work to judge good/bad artists, but rather to look at the characteristics of this work on its own. One bad painting is not indicative of a bad artist, and failed games and ideas are an expected and necessary part of the design process for a game developer. One bad function is not indicative of a bad software engineer. However, all too often, we use that metric when deciding who is competent in the field and who is not.

Unit testing or similar types of functionality verification are not critiques of the form or holistic function of a piece. In fact, simply looking for code that passes the test cases misses aspects of design, modularity, and other characteristics of software that many argue create better software products. By instituting a critique of the whole that focuses on the work and the process, we can point out elements of good design and reward holistically based on those design decisions as opposed to awarding points for correct inputs and outputs. By discussing the concept, meaning, and interpretation of the work as a created artifact, we can reinforce the nature of computing as a creative field.

The Moral of Our Story

In computing education, we recognize the key to capturing interest happens well before the college classroom. The President of the United States has recently called for CS4All (Smith, 2016). Code.org has been encouraging everyone to try an hour of code (Code.org, 2015). Other initiatives are aimed at reaching this younger audience (Kumar, 2014), and states are jumping in to give opportunities for their students (Guzdial & Ericson, 2012). Universities are forming better understandings as to the barriers presented by middle- and high-school experiences and have formulated specific treatments to address such issues (Fisher & Margolis, 2002).

However, despite these initiatives, we see that we still struggle to hold the imagination of many students. This is reflected by continued underrepresentation with Advance Placement scores (Ericson, 2014) and continuing struggles by universities to attract and retain students within the discipline and related STEM fields (Denner, Werner, & O'Connor, 2015; Martinez, Ortiz, & Sriraman, 2015; Robinson, McGee, Bentley, Houston, & Botchway, 2016). Perhaps embracing the culture of creativity, practice, and engagement is the key to unlocking the mystery.

We see a further step along this path with the educational techniques defined in the arts. When children learn to draw, they go through discrete, formed stages: **Kinematic** (scribbling), **Pre-Schematic,** involving single elements or patterns, **Schematic** (~6 years old) in which they present concepts both diagrammatic and/or narrative, **Transitional Realism** (8–10 years old) in which they begin to produce work that meets adult standards, and **Pseudo-Naturalistic/Realism** (~12 years old) that begins to incorporate advanced perspective and other organizational techniques. Interestingly, most children decide by the age of 14–16, if not earlier, whether they see themselves as artistic (Saunders, 1983). Prior to college and prior to specialized art programs. We do not yet have this type of systemic understanding of how students learn to think computationally, but it would seem to be the case that modifying this schema could provide us a direction.

All of this focus on creativity and individual expression should not, however, detract from the fundamentals of computing that we should teach our students. There are plenty of "great ideas" in computer science! In a comprehensive education, one should be exposed to them and learn them, and we would argue that all students, not just computing students, would benefit. However, the idea that pumping these ideas into their heads first and expecting them to synthesize them and use them in a miraculously advanced fashion seems backwards. We aren't giving them the tools and we aren't leveraging their passion, curiosity, and eagerness at the beginning; we're (purposefully?) creating a hurdle and then we're only interested in the runners than jump over it. To put it in terms of another discipline, sports, ultimately, we need the win, and that means we have to care about the game. When you meet students at their passion, you win. □

Reference

Code.org. (2015). Hour of code; *https://hourofcode.com/us* .

Cooper, S., Dann,W., & Pausch, R. (2000). *Alice: A 3-D tool for introductory programming concepts.* Paper presented at the Fifth Annual CCSC Northeastern Conference (pp. 107–116). Consortium for Computing Sciences in Colleges.

Daly, C. (1999). *RoboProf and an introductory computer programming course.* Paper presented at the Fourth Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '99) (pp. 155–158). New York: ACM.

Denner, J., Werner, L., & O'Connor, L. (2015). Women in community college: Factors related to intentions to pursue computer science. *NASPA Journal About Women in Higher Education, 8*(2), 156–171.

de Souza, D. M., Maldonado, J. C., & Barbosa, E. F. (2011). *ProgTest: An environment for the submission and evaluation of programming assignments based on testing activities.* Paper presented at the 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T) (pp. 1–10).

Ericson, B. (2014, January 1). Detailed AP CS 2013 results: Unfortunately much the same [Blog post]. *Computing Education Blog; https://computinged.wordpress.com/2014/01/01/detailed-ap-cs-2013-results-unfortunately-much-the-same/,2014* .

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bull., 34*(2), 79–83.

Furger. R. (1998). *Does Jane compute?* New York: Warner Books.

Garner, S., Haden, P., & Robins, A. (2005). *My program is correct but it doesn't run: A preliminary investigation of novice programmers' problems.* Paper presented at the Seventh Australasian Conference on Computing Education, Volume 42 (ACE '05) (pp. 173–180). Australian Computer Society.

Guzdial, M., & Ericson, B. (2012). Georgia computes! An alliance to broaden participation across the state of Georgia. *ACM Inroads, 3*(4), 86–89.

Kölling, M. (2010). The *Greenfoot* programming environment. *ACM Transactions on Computing Education (TOCE), 10*(4), p. 14.

Kumar, A. N. (2005). Generation of problems, answers, grade, and feedback—case study of a fully automated tutor. *Journal of Educational Resources in Computing, 5*(3).

Kumar, D. (2014). Digital playgrounds for early computing education. *ACM Inroads, 5*(1), 20–21.

Marrero, W., & Settle, A. (2005). *Testing first: Emphasizing testing in early programming courses.* Paper presented at the Tenth Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05) (pp. 4–8). New York: ACM.

Martinez Ortiz, A., & Sriraman, V. (2015). Exploring faculty insights into why undergraduate college students leave STEM fields of study: A three-part organizational self-study. *American Journal of Engineering Education (AJEE), 6*(1), 43–60.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Robinson, W. H., McGee, E. O., Bentley, L. C., Houston, S. L., & Botchway, P. K. (2016). Addressing negative racial and gendered experiences that discourage academic careers in engineering. *Computing in Science & Engineering, 18*(2), 29–39.

Rodrigo, M. M. T., & Baker, R. S. J. D. (2009). *Coarse-grained detection of student frustration in an introductory programming course.* Paper presented at the Fifth International Workshop on Computing Education Research Workshop (ICER '09) (pp. 75-80). New York: ACM.

Saunders, R. J. (1983). Reviewed work: Creative and mental growth, by V. Lowenfeld, & W. L. Brittain. *Studies in Art Education, 24*(2), 140–142.

Scragg, G., & Smith, J. (1998). *A study of barriers to women in undergraduate computer science.* Paper presented at the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98) (pp. 82–86). New York: ACM.

Smith, M. (2016). *Computer science for all; https://www.whitehouse.gov/blog/2016/01/30/computer-science-all* .

Williams, G. C., Bialac, R., & Liu, Y. (2006). Using online self-assessment in introductory programming classes. *Journal of Computing Sciences in Colleges, 22*(2), 115–122.

Wilson, B. C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education, 12*(1–2), 141–164.

Zorn, C., Wingrave, C. A., Charbonneau, E., & LaViola Jr., J. J. (2013). *Exploring Minecraft as a conduit for increasing interest in programming.* Paper presented at Foundations of Digital Games 2013 (pp. 352–359).