Updating Introductory Computer Science with Creative Computation

Dianna Xu
Bryn Mawr College
Bryn Mawr, PA
USA
dxu@brynmawr.edu

Ursula Wolz
Bennington College
North Bennington, VT
USA
ursulawolz@bennington.edu

Deepak Kumar
Bryn Mawr College
Bryn Mawr, PA
USA
dkumar@brynmawr.edu

Ira Greenberg
Southern Methodist University
Dallas, TX
USA
igreenberg@smu.edu

ABSTRACT

This paper[1] reports on the results of a multi-year project in which we identified essential pedagogy and curriculum for teaching introductory computing courses focused on Creative Computation using Processing. The curriculum aligns with a traditional 'CS1' approach as well as 'AP CS A', and goes well beyond 'CS Principles' standards to teach foundations of computer science and programming. We addressed the bridge between high school and entry-level college curriculum in computer science (American freshman high school to freshman college) and demonstrated how algorithmic art provides a powerful vehicle for diverse student populations within a broad range of pedagogical frameworks ranging from traditional structured classrooms to inquiry-based student-driven project labs. A secondary result is that instructors require long-term engagement with mentors to extend their own knowledge of computing, visual arts and appropriate pedagogy.

CCS CONCEPTS

• Social and professional topics • Professional topcs→Computing Education

KEYWORDS

CS AP; K-12 Computing; CS0; CS1; creative computation; Processing.

1 INTRODUCTION

Creative Computation via Processing[2][4,5] is a curriculum and pedagogy that provides contextualized computing through which a diverse student population can be introduced to foundational principles of computer science and develop novice expertise as programmers. Creative Computation is an emerging discipline that combines theory and methodology from computer science and engineering with aesthetic principles, creative practice and pedagogical approaches from the fine and graphic arts. As a highly interdisciplinary approach [3], it is an example of contextualized computing that can serve to attract a diverse student population to computer science. It draws from prior work in media computation [6], robots [8,10], games/animation [11,13], and music [2].

Our goal was to strengthen formative/introductory CS education in high school and college by catalyzing excitement, creativity, and innovation. The digital representation of data, access to authentic sources of big data, and creative visualization techniques are revolutionizing intellectual inquiry in many disciplines, including the arts, social sciences, and humanities. We assert that CS1 should be updated with contemporary, diverse examples of computing in a modern context. This paper reports on the success of adapting a curriculum and pedagogy, namely a traditional CS1 scope and sequence, with an art studio pedagogy in which students engage in highly individual artistic design. This paper summarizes the qualitative evidence to date that Creative Computation can be successfully integrated into a variety of high school and undergraduate introductory courses.

Contextualized computing is increasingly being introduced into American grades 6-11, articulating with the Common Core Standards in Language Art. With few exceptions such as [12], the primary emphasis is on computing principles applied to a sampling of disciplines rather than immersion in an interdisciplinary domain. Using a qualitative research approach we studied the degree to which an established introductory curriculum could be disseminated to high school and diverse undergraduate institutions with a broad range of pedagogical perspectives.

Through short-duration workshops and mini-grants, several school instructors have adapted our Creative Computation curriculum to courses including High School Pre-AP Computer Science, AP CS Principles, AP CS A, High School Post-AP, Community College Computer Science, Community College Media Studies, Computer Science 0 (CS0), and Computer Science 1 (CS1). The institutions ranged from small private elite high schools and colleges, to large urban public high schools and universities, with course structure including teacher-centric instruction and inquiry-based labs. The curriculum was successfully adapted as modules in semester or yearlong courses, as full courses that ranged from quarter, to tri, to semester and full year duration, and one instance adapted the curriculum for an online course for both high school students and undergraduates.

SIGCSE '18, February 21–24, 2018, Baltimore , MD, USA © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159539

[2] Processing.org

Reports from participant instructors in the workshops and mini-grant program suggest that Creative Computation was universally successful. This can be attributed to the objective to create a flexible curriculum and pedagogy that can be adapted by instructors, rather than requiring adoption of a highly structured scope and sequence. This would seem obvious with the objective of promoting contextualization. The remainder of this paper documents how a traditional CS1 over-arching structure, that emphasizes individual student creativity in the arts, can meet the needs of a diverse constituency and provide meaningful contextualization.

A secondary outcome was to provide further evidence that the current popular model of short duration workshops with online resources is insufficient for developing instructor expertise. Contextualized computing in general and creative computation in particular require interdisciplinary expertise. Providing sufficient resources for such interdisciplinary professional development remains an open question. The workshop and mini-grant participants all requested more extended engagement in both in-person and internet-based venues. We propose recommendations for next-steps in adequately supporting professional development specifically in Creative Computation in CS1, and for contextualized computing in general.

2 CREATIVE COMPUTATION FOR CS1

Creative Computation showcases the theory of visual arts rendered through computer code. In addition to a firm grounding in programming principles, students learn essential graphics operations such as transformations, iteration and randomization, algorithmic drawing, animations, basic physics-based simulations, and interactivity. Domains of inquiry go well beyond commercial level media graphics (e.g. what you might see on a web-page or mobile device) and include creative use of image processing and emergent systems. As a true blending of science and art, Creative Computation requires understanding of and practice in the intersection of analysis and aesthetics. For example, pixel-based image manipulations require mastery of efficient algorithms for data structures such as two and higher-dimensional arrays; text and data visualization require a fusion of artistic design and application of sophisticated data structures and algorithms; rendering L-systems requires deep understanding of recursion and recursive structures.

Creative Computation [4,5] requires students to engage in artistic creative acts to practice and master essential concepts, not only in media arts, but in computational thinking, algorithm design and programming. Used as a vehicle for teaching introductory computer science, our approach required blending opportunities for individual artistic expression with a traditional Java-based CS1 scope and sequence. The curriculum follows a standard CS1 course, but instead of solving for roots of polynomials, simulating gas station/cash registers, or implementing well-known algorithms and techniques, our labs provide avenues for highly individual creative expressions to produce artwork, interactive multimedia, data visualizations, and games.

2.1 Scope and Sequence

The CS1 scope and sequence shown in Table 1 is a variation of an *Objects Gently* approach to teaching programming with Java [7]. Object-oriented concepts are introduced neither at the beginning or the end of the semester.

Programming Concepts	Creative Computation Technique
What is programming	Built-in drawing functions; primitive shapes, coordinate systems, creative coding principles
Variables, data types, expression	Drawing in scale, proportional distance and size to a vanishing horizon specified by a y-variable
Mouse/keyboard interactions	Shapes drawn via mouse clicks
Control structures: loops and conditionals	Use of simple iterations to draw many shapes Use of conditionals in simple simulations: ball drop, bouncing ball
Writing functions with control structures, randomization	Parameterized shape drawing: location, size, color etc.; Randomize display
Simple objects, super and subclasses, object instantiation vs class definition	Shapes modeled as objects: data fields store parameters, methods implement rendering and animation, dynamic instances generated by mouse click
One dimensional arrays	Complex scene with many shapes(objects) stored in arrays
Math applications: polar coordinates, trigonometry, rotational geometry	Mandala art, complex abstract/geometric design, composition of polygons, stars and line strings, creative aesthetics
OOP principles, inheritance, abstract classes and interfaces	Collaborative project with generic object types, instructor provides interface, students contribute sub classes
Manipulating two dimensional arrays	Image processing, Game of Life, tiling and tessellation
Recursion, recursive functions	Procedural art (e.g. fractals), L-systems
Data sets, data mining, data visualization, strings and files	Visually manipulate string Visualization techniques

Table 1: Creative Computation Scope and Sequence

Exercises consist of a sequence of programming projects in which students individually or collaboratively construct a work of art, ranging from still graphics as shown in Figure 1, to data visualizations and interactive games. For example, to teach object inheritance, each student is asked to create a subclass of a sea-creature interface/abstract class. The hierarchy is combined into a single project instantiating the contributed subclasses in a creature aquarium. Every exercise has a well-defined basic part and offers students the opportunities to explore open-ended creative projects. Students take ownership

of their work. As a consequence, no two submitted projects are the same, and plagiarism is visually obvious.

2.2 Creative Computation in Processing

Our approach follows in the footsteps of John Maeda, who is both a formally trained artist and computer scientist. Inventor of the programming environment Design by Numbers [9], a precursor to Processing, he pioneered an approach to "creative coding" that radically *re-contextualized* computer code–from an applied math notation to a creative medium, on par with charcoal, paint, clay, etc. Creative coding is an exploratory and aesthetically driven approach, where students build visual designs and artworks iteratively as they expand their projects.

Our curriculum assumes use of the Processing IDE. It should be stressed that this is not a curriculum to learn Processing, but rather a traditional CS1 – introduction to programming, problem solving and foundations of computer science utilizing the rich and friendly graphics library of Processing. It can be taught in any programming language with a mature graphics API.

Processing is built on top of Java with a simplified syntax and an API that focuses on graphics/media programming. It is a robust and full-featured language that has use both in the classroom and beyond; it is used widely in industry and growing quickly in popularity. It supports code writing in Java, Python and JavaScript. A Processing project is developed in Java or Python in a simple IDE that dispenses with the complexity of creating a complex object structure in Java or mastering library imports in Python. Within the last two years P5.js[3] provides a JavaScript web-supporting framework in which to write Processing functions using JavaScript syntax. Our CS1 curriculum is implemented entirely in the Java version to align with the AP CS A and traditional Java based curricula. Sample code in our repository is entirely in Java, however lab assignments can be adapted to either JavaScript or Python to accomplish the learning goals stated in the materials. We also have a successful adaption of our curriculum in C++ with openFrameworks[4].

3 CURRICULUM ADAPTION

The efficacy of the Creative Computation approach comes from a series of opportunities to experiment with how the curriculum as a whole, and in part, could be adapted by a range of institutions. Through initial deployment in two undergraduate institutions and two high schools, a series of short-duration workshops introduced educators to the curriculum and pedagogy. Through mini-grants, nine institutions adapted the curriculum and provided qualitative analysis of results to an external evaluator and are summarized here.

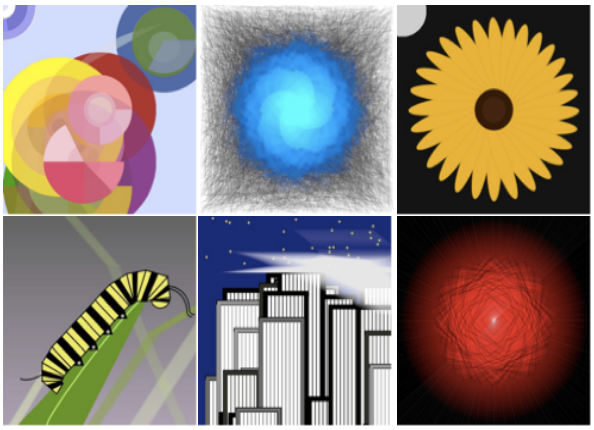

Figure 1: Student artwork

3.1 Initial Undergraduate Implementation

The curriculum outlined above has been in place within undergraduate introductory courses at Bryn Mawr College, Bryn Mawr, PA and Southern Methodist University, Dallas, TX for the past eight years.

Bryn Mawr Collge (BMC) is a small, all-women's liberal arts college in the suburbs of Philadelphia that enrolls 1,300 undergraduates. BMC teaches and values critical, creative and independent habits of thought and expression in an undergraduate liberal arts curriculum for women. The Computer Science Department offers a B.A. in Computer Science as well as a minor in Computer Science and in Computational Methods.

Southern Methodist University (SMU) is a private university of 11,000 students (6,000 undergraduates) near the vibrant heart of Dallas, TX. SMU offers nationally competitive undergraduate and graduate CS and Engineering programs to a larger and more conventional student body. The Creative Computation Program is supported through both the School of the Arts and the Department of Computer Science and Engineering housed in the School of Engineering and offers degrees in both Computer Science and Computer Engineering.

In both institutions Creative Computation was developed and adapted to fit the local intellectual culture. Over a dozen different instructors not associated with the grant have successfully offered our curriculum more than twenty times. See www.cs.brynmawr.edu/visual for more information.

3.2 Initial High School Implementation

As a curriculum intended to support the bridge from high school to college, a natural consequence was to investigate how it could be adapted to the cultures of high school. Two high schools were initially selected [14]. The courses were taught in these two schools that are at the extreme end of the spectrum of high school cultures, using pedagogy that is similar in

[3] P5js.org P5js.org
[4] openframeworks.cc/

the emphasis on project work, but markedly different in use of materials and presentation style. Sidwell Friends School (SFS) is a private coeducational PK-12 Quaker school in Washington, DC, in which computer science is taught entirely through a project-based lab with heavy emphasis on student-centered inquiry. James Martin High School (MHS) is a large urban public school in Texas that enrolls over 3300 students in grades 9-12, where computer science instruction models traditional high school mathematics instruction. As a large public high school that accommodates a diverse population with a full range of socio-economic needs, expectations for achievement require significant reinforcement via extensive use of highly structured materials. The computer science program at MHS is targeted toward Advanced Placement and formal lecture is reinforced through worksheets and exercises that lay the groundwork for structured project assignments. The participant teachers in both schools successfully adapted the college-level creative computation model to the following courses: as in [14]:

- An introduction to computer science course that starts with 4 weeks of Python following by Creative Computation with Processing. The course covers materials traditionally associated with CS1 with an introduction to classes but without inheritance.
- An introduction to computer science course that uses Creative Computation with Processing exclusively and covers material traditionally associated with CS0.
- An advanced computer science course that parallel expectations for a traditional CS1/CS2 experience exclusively taught with Creative Computation in Processing.
- An AP CS course that requires students to sit for the AP exam. Creative Computation projects are used intermittently because the test is not contextualized to the arts, but uses draws from a broad spectrum of applications.
- An Artificial Intelligence and Game Design course that has evolved into a significant example of Creative Computation that goes beyond the fine arts.
- An advanced computer science class that covers material traditionally found in CS2 as well as providing opportunities for kinesthetic game design.

3.3 Summer Workshops

In 2014, a four-day summer workshop was held at Bryn Mawr College[5]. There were fifteen participants: seven high school teachers and eight college faculty. Five public and two private high schools were represented including a leading K-12 school for students with learning disabilities. The college faculty included a large research university, two small private liberal arts colleges, as well as a public university and two community colleges. Feedback from participants was overwhelmingly positive. Five contributed projects to our repository, and all but one indicated plans to adapt the curriculum into their programs.

In 2015, we held a second four-day summer workshop at Soutern Methodist University. Despite a budget limited to twenty participants, thirty-three people were accepted into the workshop; twenty-eight attended. Of those twenty-eight, twenty-one were faculty members: twelve high school teachers and nine professors from colleges and universities around the country. Among the teachers, nine were from public schools, and three from private schools. The college faculty members ranged from a well-known research university, a small private liberal arts college, three public universities, and an Ivy League university. There were also non-faculty participants: tech industry and non-profit organizations that run extracurricular and youth summer camps for computing. All of the participants either indicated intention to adopt the Creative Computation approach, or were already teaching with Processing in some form and intended to adapt our curricular materials and approaches.

3.4 Mini-grant Institutions

Between 2014 and 2017 instructors from ten different institutions were awarded mini-grants to adapt and implement the Creative Computing curriculum in their own classrooms. Nine of these filed final reports summarizing successful implementation, one initial grantee withdrew before implementation, due to local institutional staffing and course scheduling difficulties. Additionally, an external evaluator carried out pre-post interviews and student surveys on all these courses.

The curriculum was implemented at four high schools (2 public, 2 private), a public 2-year community college, a private 4-year liberal arts college and three 4-year universities (2 public, one private). One college course was offered online for both high school and college students. The four major regions, as defined by the US Census, were represented: one each in the Northeast and South (both high school), three in the West (two high school, one undergraduate) and four in the Midwest (four undergraduate, one on-line allowing high school student to register).

The courses that were offered spanned a full range of traditional levels in computer science and in introductory media arts. Introductory computer science is comparatively new and lacks a consensus on foundation.[6] As a metric for evaluating the efficacy of Creative Computation we use the following generally accepted curricular foci to categorize the courses taught in the mini-grant program.

[5] http://cs.brynmawr.edu/visual/bmcworkshop.html.

[6] See http://www.acm.org/education/curricula-recommendations for CS0 and CS1, http://www.exploringcs.org/ for Pre-AP HS – e.g Exploring Computer Science, https://apcentral.collegeboard.org/courses for AP CS A and AP CS Principles. We have not found a professional society standard for Media Arts programming.

- Pre-AP High School: As exemplified by the nationally recognized *Exploring Computer Science.*
- AP CS principles/CS0: CS0 is considered a foundations course for non-majors or entering college CS majors with no background in computer science. The AP CS Principles course is intended to articulate with that college level offering.
- AP CS A/CS1: The traditional computer science course for entering CS majors.
- Media Arts Coding: A course offered in Digital Arts programs that introduces programming.

The high school courses were titled: AP CS A, AP CS principles, Introduction to Programming (Pre-AP), Pre-AP CS A (three instances).

The undergraduate level courses were titled: Applied Logic (A prerequisite for CS1, not a CS0), Beyond Photoshop (A CS0 course), Critical Games (A media arts course), CS1.

Implementation ranged from modules (in the AP CS A courses) to full semester adaption of the creative arts projects curriculum outlined in section 2 for courses at the pre-AP and AP Principles High School level, and for all of the undergraduate courses as pre-CS1 (e.g. Logic and CS0), CS1, and for Media Arts.

Our initial efforts at both the college and especially the high school level identified a broad spectrum of pedagogical styles from highly structured to intensively inquiry-based. The mini-grant offerings, not surprisingly fell in the middle without clear metrics for distinguishing between them. They all did however include the following:

- Short lecture and resource materials to introduce programming (e.g. variables, control structures) and creative arts concepts.
- Practice via short-guided assignments that allowed for individual creativity.
- Opportunities for large project work with a focus chosen by the student.

The remaining sections summarize outcomes and recommend changes for curricular and teacher professional development.

4 OUTCOMES

The major expected outcome of this work was that Creative Computation as an adapted curriculum can successfully meet a broad range of curricular goals at the bridge between high school and introductory college computer science. A key element is the emphasis on project work in which students can express artistic creativity and create an individual agenda for project goals. This project also demonstrated that the traditional CS1/CS AP A curriculum structure is viable as a vehicle for introductory computing, provided there is contextualization, in this case media arts. Quotes from mini-grant participants provide insight:

> *We have been teaching [CS 0] for 5 years. This was by far the most engaged group of students we have worked with; the combination of technical rigor, criticality, and contemporary themes/materials/topics seems to have inspired and encouraged an unusually open and responsive group dynamic.*
>
> *These creative tasks allowed for both computational and "artistic" success. Students could produce something that was simplistic in terms of code, but successful in terms of humor or emotional appeal. This allowed for a wonderful classroom climate in which students of all levels felt capable of producing work they would be proud to share with classmates.*

A hoped for, but not necessarily expected outcome, was that the short-duration workshop that exposed participants to a studio art experience would support adaptation of our pedagogy to their own needs. Evidence to that effect is that all of the mini-grant participants reported that they successfully completed their course and were going to continue using Creative Computation. Consider this example:

> *Their projects and the process was a success, and one that I intend to replicate after the AP test. The entire class became more collaborative and kids were motivated not from a grade, but from seeing something on the screen that they wanted to replicate. In addition, it has been rare for students who do not know each other to enter into a discussion. With this project, many were looking and asking how to take their own projects to the next level. Many kids who were checked out for most of the year, basically came back to write more code than they did the entire year, and they were doing it based on their own purpose-driven initiatives. I could hardly keep them out of the computer lab. I would like to use this more next year in all my classes, but in particular, AP Computer science just because many kids find the current grind a bit more than they care to take.*

Unexpected outcomes did, of course, occur. The commitment to NSF was to produce a workshop model and repository of material resources. This proved inadequate. Each mini-grant participant expressed a need for more mentorship with experts to expand their computing and artistic skills, to brainstorm pedagogy, and to identify curricular connections. An attempt to create an online community of our own during the mini-grant course execution period did not generate sufficient traffic to engage the group. However, all participants did articulate goals to pursue Creative Computation further through existing resources.

We address the question of what merits claiming success. In this paper we do not list statistics of diversity, nor increased enrollments, nor attitudinal outcomes. Seven years ago when this project started, increasing enrollment was seen as a bell-weather of success. But interest in computer science has exploded with enrollments stretching teaching resources in both high school and undergraduate programs. Consequently, there is no statistically valid correlation between Creative Computation and increased enrollment, nor, we suggest, can any contextualized approach make that claim. Similarly statistics on who enrolls in introductory computer science classes is not correlated with successful curriculum. Schools on both sides of the bridge now

require computer science, often because of state mandates. In institutions without an explicit requirement, due to lack of teaching resources, students cannot enroll or are placed in over-crowded classrooms. A companion paper will present a methodology and conclusions on student affect. Consequently, we report here on how the curriculum was successfully used by the simple standard that teachers self-evaluated their success, critiqued their short comings and articulated what they would improve in the next round. Throughout, the reporting has been consistently positive.

5 RECOMMENDATIONS

The AP CS Principles curriculum was intentionally developed to address the shortcoming of the AP CS A – that it was inaccessible to many students, with too much emphasis on the Java programming language and not enough emphasis on core computer science principles. The Creative Computation curriculum illustrates that the depth of code development, problem solving, and principles of software design explicit in the CS1, AP CS A curricula can be taught in a manner that engages students. Programming is essential to learning computer science, but the coding must be purposeful. In the curriculum presented here, that purposefulness comes from the emphasis on creative arts. This speaks to a need for reconsideration of *how* foundations of programming are introduced rather than *depth of knowledge* of those foundations. While we promote creative art as a vehicle for the *how,* this is certainly not the only framework, but we do assert that the *depth of knowledge* should not be minimized. Programming, problem solving, and software design are skills that, like any other skill, need to be practiced and mentored. Consequently, our recommendation on both sides of the bridge is to keep the depth of knowledge, but to identify how standard curricula can be *adapted* to the local goals, objectives and projected outcomes.

Simple adoption of curriculum requires teacher professional development. Adapting curriculum to local goals requires expertise in both the subject area and the pedagogy. This project illustrated that minimal training via short duration workshop and static online materials was sufficient to bootstrap adaption. But a significant outcome of the mini-grant program was the hunger among all the participants for more – of everything. A compelling question for the computer science community is how to provide those resources in a way that engages a community – that provides the kind of art studio approach we promoted to the very hard task of teacher professional development.

ACKNOWLEDGMENTS

This project is supported in part by grants from The National Science Foundation (DUE-0942626, DUE-0942628, DUE-1323463, DUE-1323305, CCF-0939370, and CCF-1140489).

REFERENCES

[1] Jessica D. Bayless and Sean Strout. Games as a "Flavor" of CS1. In *proceedings of SIGCSE 2006*. ACM Press 2006.

[2] Robert E. Beck, Jennifer Burg, Jesse M. Heines, and Bill Manaris. Computing and Music: A Spectrum of Sound. *Special Session, SIGCSE 2011*. Dallas, TX, March 2011.

[3] Cassel, L. and Wolz, U Interdisciplinary Computing, Successes and Challenges. In *Proceedings of SIGCSE 2013*. ACM Press 2013.

[4] Ira Greenberg, Deepak Kumar and Dianna Xu. Creative Coding and Visual Portfolio for CS1. In *Proceedings of SIGCSE 2012*. ACM Press 2012.

[5] Ira Greenberg, Dianna Xu, and Deepak Kumar. *Creative Coding and Generative Art in Processing 2.0*. friends Of ed/Apress 2013.

[6] Mark Guzdial. Introduction to computing and programming with Python: A Multimedia Approach. Prentice-Hall, 2004.

[7] Elliot Koffman, E. and Ursula Wolz, CS1 using Java language features gently. In *Proceedings of SIGCSE/ITiCSE 1999*. ACM Press 1999.

[8] Deepak Kumar, Doug Blank, Tucker Balch, Keith O'Hara, Mark Guzdial, Stewart Tansley, Engaging Computing Students with AI and Robotics. *Symposium on Using AI to Motivate Greater Participation in Computer Science*, 2008.

[9] John Maeda *Design by Numbers*, MIT Press 2001.

[10] Jay Summet, Deepak Kumar, Keith O'Hara, Daniel Walker, Lijun Ni, Doug Blank, and Tucker Balch. Personalizing CS1 with Robots. In *Proceedings of ACM SIGCSE 2009*. ACM Press 2009.

[11] Ursula Wolz, Christopher Ault and Teresa Nakra, "Teaching Game Design through Cross-Disciplinary Content and Individualized Student Deliverables", *The Journal of Game Development*, adapted based on invitation from presentation at the 2nd Annual Microsoft Academic Days Conference on Game Development , February 22 - 25, 2007

[12] Ursula Wolz, Kim Pearson, S.Monisha Pulimood, Meredith Stone, Mary Switzer. Computational Thinking and Expository Writing in the Middle School: A novel approach to broadening participation in computing, *Transactions on Computing Education*, 2011, Volume 2, article 9.

[13] Dianna Xu, Douglas Blank, and Deepak Kumar. Games, Robots and Robot Games: Complementary Contexts for Introductory Computing Education. In *Proceedings of Third International Conference on Game Development in Computer Science Education* (GDCSE'08), 2008.

[14] Dianna Xu, Aaron Cadle, Darby Thompson, Ursula Wolz, Ira Greenberg, and Deepak Kumar. 2016. Creative Computation in High School. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 273-278. DOI: https://doi.org/10.1145/2839509.2844611