Collaboration and Computational Thinking

A Classroom Structure

Benjamin Worrell
Carson Middle School
Colorado Springs, US
bworrell@ffc8.org

Catharine Brand
Science Discovery Program
University of Colorado, Boulder
Boulder, USA
catharine.brand@gmail.com

Alexander Repenning
Department of Computer Science
University of Colorado, Boulder
Boulder, USA
ralex@cs.colorado.edu

Abstract--Building a classroom environment based on student-led, meaningful collaboration is an ideal being promoted by educators and administrators seeking to equip students with 21st century workforce skills. More and more often, teachers are challenged to design lessons that utilize students' innate desire to communicate with one another rather than more traditional direct instruction. With a heavy focus on student-driven instruction and a reduced role for direct instruction, Carson Middle School's Game Design I course is able to utilize collaboration as a means for allowing students to not only learn but master and retain Computational Thinking Patterns and apply them in formal summative assessments. Through the collection of these data points throughout the course of a semester of learning, an educator can fully appreciate the learning of Computational Thinking skills that occurs as students build a sequence of games and simulations.

Keywords— computational thinking; computational thinking patterns; assessment; collaborative programming; middle school computer education, game design

I. INTRODUCTION

Computers have become ubiquitous in the 21st century workplace. In pockets and purses, many individuals carry powerful computers which incidentally are capable of making calls over the cell phone networks. When Jeanette Wing made her argument in a 2006 CACM article that computational thinking is as fundamental a skill as reading, writing and arithmetic for children, her message was like a stone thrown into a pond. The ripples are still spreading as computer science researchers and educators debate the definition of computational thinking and what computational thinking should look like in a K-12 classroom. In her article [9], Wing defined computational thinking as "solving problems, designing systems and understanding human behavior, by drawing on the concepts fundamental to computer science". In 2011, Barr and Stephenson [1] published a list of activities that students could actually do in a classroom in order to learn computational thinking skills. Of particular interest, the recommendations from their panel of experts in computer science and education included the following items: designing solutions to problems, implementing designs by programming, testing, debugging, modeling, learning the appropriate vocabulary, recognizing the abstractions and group problem solving.

For a number of years, the Scalable Game Design (SGD) research group has trained and supported middle school teachers who teach game design to their students as a method for increasing the students' motivation to learn about technology [6]. In the long run, we hope that some of the middle school students, including girls and members of underrepresented groups, will pursue their interest in technology by taking high school and college courses leading to careers in the tech industry. In addition to motivating learners by letting them build colorful and creative games on computers in a few hours, the game design activities in AgentSheets and AgentCubes, the SGD end-user 2D and 3D programming tools [6], also serve to introduce these middle school students to computational thinking. Ideally, students in classrooms with SGD teachers build a sequence of games and then use what they have learned about design and programming in AgentSheets and AgentCubes to create at least one science simulation.

The SGD researchers have looked for approaches that would enable teachers and students to learn how to program more easily in our programming environments. We have found one effective tool in Computational Thinking Patterns (CTPs), which are abstractions of mid-level constructs that encode a single agent behavior. We have identified an initial set of patterns general enough to appear in both games and science simulations and we continue to add CTPs as we observe new learning activities in classrooms and adapt our programs to run on new platforms. Our CTPs consist of several program primitives (conditions and actions), arranged in the if-then rules that characterize the AgentSheets/Cubes programming environments. Some computational thinking patterns include methods containing rules that accomplish specific behaviors. AgentSheets programs can be automatically analyzed to show which CTPs they contain and each CTP can be evaluated to show whether it's implementation is complete [3].

Polling is an example of a complex Computational Thinking Pattern. Polling allows an agent, usually a controller agent dedicated to that purpose, to keep track of the number of agents of a particular type. For example, imagine a game that ends when the hero has collected all the treasures. In this game, the controller agent's main method sets a global variable to 0, asks the Treasure agents to count themselves and makes the controller do it's own check-win method. The controller's check-win method shows the player a message and stops the

game when the global variable remains at 0 because no Treasure agents remain in the game. Each Treasure agent also contains a method to implement the polling CTP which adds 1 to the global variable that holds the count. The polling CTP can be modified to make PacMan win the game when he has consumed all the pellets or added to an ecosystem simulation to make it stop running if all the top predators disappear. When the students copy, paste and modify the polling CTP in a new program, this process can serve as a step towards complete understanding of the CTP [8].

At the SGD teacher workshop, SGD researchers encourage participating teachers to use Computational Thinking Patterns as they explain desired program behavior to students. This advice contrasts with the work of other researchers who have identified patterns in student-created programs as a means of evaluating their degree of programming sophistication [5, 7] but have not used them as teaching aids. In the SGD classroom, CTPs provide a means for describing solutions without going to the level of detail that includes every programming primitive involved. Having a mid-level description of the solution enables novice programmers to program more efficiently [8], although they are still challenged by figuring out exactly which primitives to include in each CTP, how to edit a CTP to achieve an agent behavior in a new setting and how to combine CTPs to create an appealing game. For example, PacMan requires the absorption, diffusion and hill climbing and collision CTPs (see [2] for more complete descriptions of these CTPs). The students employ several CTPs in each game they build. After creating a sequence of three or four games, learners have been exposed to a dozen CTPs and the class has developed a shared vocabulary for discussing program design and functionality.

Assessment plays an important role in a Scalable Game Design classroom. Are the students learning what the teacher has taught? It is not sufficient for a class to have a couple of programming star students who really "get it". How does the teacher know that every student took something away from the weeks of work? SGD teachers who have taught game design with Computational Thinking Patterns can design assessments to see whether students can identify CTPs. Another possibility is to test the students on using the CTPs in a novel setting like an unfamiliar game or a simulation. Assessment can provide guidance to the teacher on how to refine teaching strategies as well as justify the time and energy spent on game design to the school administration. Assessments should also be simple to implement and evaluate by a single teacher.

II. Teaching Game Design Collaboratively

The lead author, who has a background in studio art, teaches Game Design electives to 6th, 7th and 8th graders at a middle school (CMS), which is located on a military base in Colorado. The pleasant modern middle school building has computer labs that provide one desktop computer per student to classes of up to 25 students. Many of the ethnically diverse students are the children of military families and have attended a number of different schools of varying quality. Due to the seemingly spontaneous deployment schedule of their parents, this middle school has a highly transient population of students. Indeed, the revolving door of CMS means classrooms can add or lose multiple students over the course of a single semester. This is both a strength and weakness to the classroom environment as students are quick to make friends but slow to build relationships. The amount of trust it takes to accept instruction and criticism from a peer is large and is a struggle for students who may be staying at the school for only a few months. For these students to flourish, the school must supply a supportive and understanding atmosphere. However, these children do bring a unique advantage to the classroom. They have grown up in a military culture where adults share responsibility and work together to achieve goals. They are at home with the language of collaboration and accountability.

Building a classroom environment based on student-led, meaningful collaboration is an ideal being promoted by educators and administrators seeking to equip students with 21st century workforce skills[4]. More and more often, teachers are challenged with designing lessons that utilize students' innate desire to communicate with one another rather than more traditional direct instruction. With a heavy focus on student-driven lessons and a reduced role for teacher-led coaching, CMS' Game Design I course is able to utilize collaboration as a means for allowing students to not only learn but master and retain complex computational thinking skills and apply them to formal summative assessments. Outlined in this essay is the collaborative process used at Carson Middle School to teach Computational Thinking to students in grades 6, 7, and 8. Not only does this learning process allow students to program games which utilize CTPs in varying degrees of complexity, but it also grows students' abilities to communicate with peers, analyze problems, and seek answers independent of direct teacher intervention. Also discussed are two separate assessment tools which allow the instructor to evaluate students' problem-solving and critical thinking skills as well as retention of material learned over the course of the semester. This collaborative process solves the problem of making students become active learners and enables them to retain information rather than simply copying from a previously created tutorial.

There are a few "red flags" which often occur as teachers attempt to design lessons that allow student collaboration. In a typical cohort group of four students, there will often be one or two students who take advantage of the group setting and allow other students in the group to perform the majority of the work. A teacher may solve this problem by assigning specific 'roles' to the students who are then assessed independently once the final project is completed. A second issue arises here as students who complete their part of the project become frustrated as other students turn in low-quality products or, often, nothing at all. Perhaps a solution to this issue is not to utilize group work as the assessment tool but rather as the primary form of instruction while the assessment is merely a reflection of the learning that takes place within the group. This would allow students to work together and collaborate on ideas while allowing for them to independently demonstrate what they have learned in individual assessments without having an effect on the other members of their team.

While the CMS Game Design I course utilizes AgentSheets 4.0 as the primary programming tool for all games and simulations, the collaborative process described herein could be modified and applied to many different programming tools in a variety of situations. Each game or simulation is introduced with approximately thirty minutes of direct instruction by the teacher. This instruction time is utilized primarily to introduce new concepts and CTPs and allow students to see them in action. This is the only point in time over the course of a one or two week project where the teacher will be in front of the students giving instructions. During this initial stage, students are required to experiment with the new CTPs and content in the AgentSheets program. After this, students are placed into cohort groups of 2-4 individuals. Each member of a group is given a specific task to complete in building the game or simulation required in that module. These tasks are given code names that relate to the students' involvement in military culture such as "Alpha" and "Bravo." While this may seem like a fairly small takeaway, it is important for any teacher to shape every aspect of a lesson to the audience they are serving.

An example of one such task is in the game Frogger. This project utilizes the CTPs Absorb, Generate, Transport, and Collision. Each student in a group would be assigned one of these CTPs to master. Once they have their assignments, students are then tasked with creating a Do It Yourself (DIY) Tutorial which includes photographs taken from online tutorials, original rather than copied explanations, and other important information to apply that CTP to the Frogger game. Once all students have completed their DIY Tutorials, they then each take turns as "student instructor," teaching their portion of the game to the other members of their cohort group. Over the course of several class sessions, students piece together a complete Frogger game. It is very important that the student instructor does not share the contents of their DIY Tutorial by allowing their teammates to directly copy from it. On that same note, student instructors are not allowed to directly interact with their teammates' computers. Student instructors may only point and speak as they share their information. Once the game or simulation is complete, students are evaluated on the implementation of the various CTPs according to a standardized rubric. It is important to note that throughout this process the instructor functions mostly as tech support, helping students with issues related to the computers themselves and not the programming. This process is repeated several times for a variety of games until all CTPs have been addressed and learners are ready to use them in simulations and an independent capstone project.

III. Dungeon Adventure Assessment

Once students have created specific games which feature all required CTPs for the course, they are then evaluated formally via the creation of Dungeon Adventure, a game which tasks students to utilize previously learned concepts and CTPs in new or unexpected roles. Essentially, students are given a checklist which outlines behaviors which must be included in the final game. "Hero Agent generates Arrow Agents" and "Monster Agent uses hill climbing to track Hero Agent" are written in list format. Students must then utilize prior knowledge, DIY Tutorials, and peer collaboration to construct the game. Since they only have a few hours for this assignment, these resources must be used knowledgeably and efficiently. As was the case previously, students are not allowed to directly manipulate one another's games, nor are they allowed to directly share their DIY Tutorials for the purpose of copying. The result is a challenging test of a student's ability to use their learning to construct solutions to innovative problems that they have not yet encountered. Once again, the teacher is available for tech support but does not provide any sort of assistance in programming.

Looking at the results of a semester's student group of 52

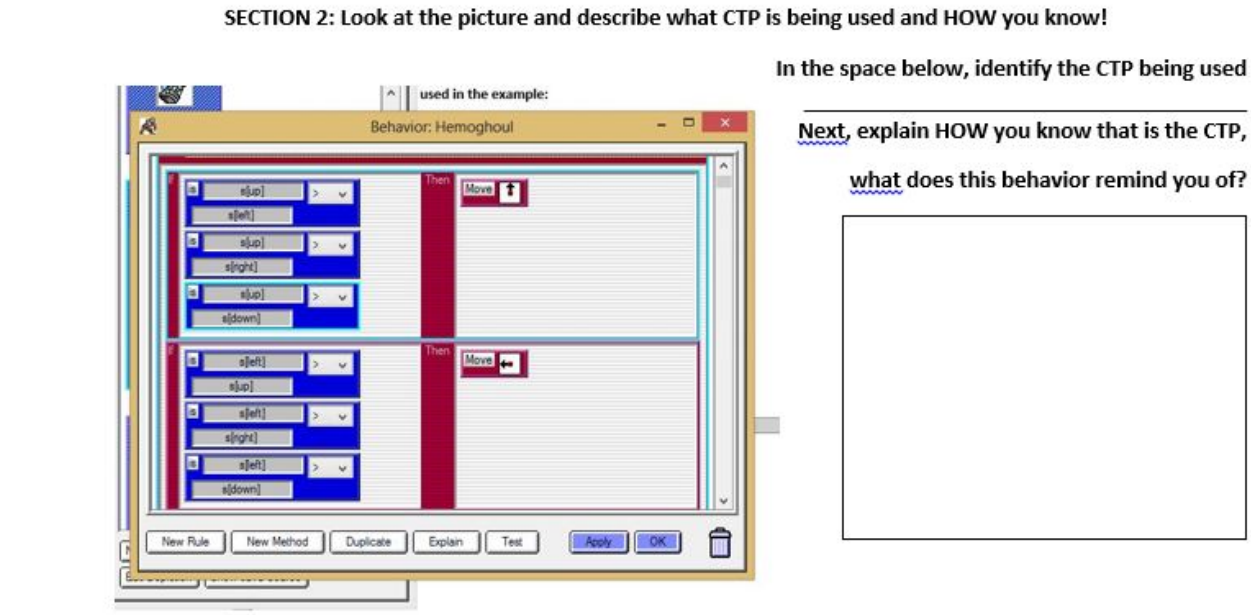

Fig. 1. CTP Identification test item.

students, 40 (77%) students successfully applied 75% or more of the objectives for this assessment to their game, 12 students (23%) successfully applied between 50% and 74% of the objectives, and no students completed less than half of the objectives. Even though students still have the ability to access outside sources to assist in the creation of this game, the objectives required for success demand students to apply their knowledge in a new or augmented manner which significantly raises the rigorous nature of the evaluation.

IV. COMPUTATIONAL THINKING PATTERNS ASSESSMENT

While Dungeon Adventure is a helpful tool for the educator to evaluate student learning in a collaborative environment, it is also important to assess a student's ability to program and recall information independently as well. Some period of time after Dungeon Adventure is assigned and graded, students are given a pencil and paper summative assessment which is completed without the use of any outside resources. A pretest was omitted because although the students may have had some exposure to gaming or programming, they have no experience with the unique content covered in this class. The summative test itself is broken up into three sections, with each section requiring students to apply the knowledge they learned in programming games but in slightly different ways. In the first section, students are asked to read several scenarios which illustrate the various CTPs students encountered over the course of their time in class. A sample item reads "A high speed police car chase ends with the culprit smashing into a fire hydrant on the side of the road. It turns out she was texting while driving and didn't realize the police were chasing her." Students must then identify these CTPs and justify their answers utilizing evidence from previous games as well as from the text. In the second section (Fig. 1), students are asked to dissect a photograph of a particular programming behavior and identify what CTPs are being used. Finally, students are given a list of desired behaviors (i.e. an agent that moves towards a destination and disappears once it arrives) and are then asked to write in the specific condition and action code blocks that would allow these behaviors to operate correctly.

The resulting data collected from the same tested group of 38 students, 43% female and 57% male, revealed encouraging results. 28 (74%) successfully completed 75% or more of the objectives for the assessment. This left 10 students who completed between 50% and 74% of the objectives and there were no students who completed less than half of the objectives. This very small decrease in overall student achievement is extremely encouraging as it suggests that the collaborative learning students experience in the class does not limit success when they are required to perform on their own.

V. DISCUSSION OF RESULTS

The first two years of the Game Design I class focused on "watch and copy" direct instruction, which educators feel is the most risk-free method of teaching high level concepts. While students completed impressive and functional programs, their projects were not unique in any way. Furthermore, students did not seem to retain the information as this method of instruction isolated each project and did not allow students to see the many connections between the different projects. Direct instruction stifled creativity, student interaction, and, ultimately, the long-term application of computational thinking which might have extended beyond the classroom.

During the first year of implementation for the CMS Game Design course, a very similar pencil and paper test was administered to students. Students were taught via step by step instructions projected on a screen and operated by the teacher. Students were expected to follow along and emulate what the teacher was doing. While most students were able to create the required game, there was a significant rise in frustration on the part of both the students and the teacher as the diverse group of students worked too fast, too slow, or just gave up altogether. Even though the games created were mostly consistent with games created by the students programming in teams this year, the results from the pencil and paper assessment were markedly lower. Students who had seemingly demonstrated an ability to program the various CTPs and behaviors required by each individual game could not identify or use the CTPs outside of their original context. This led to a significant overhaul of the curriculum for the program and eventually led to the process and evaluations described previously.

Due to the transient population at CMS, teachers cannot anticipate having students for an entire semester. The collaborative nature of the CMS Game Design course does allow students to arrive late to the course and be integrated quickly. Because instruction is student-led, there is an expectation that new students will be caught up by their peers so that they can become contributing members of their groups in as short a time as possible. Because of this structure in the classroom, students arriving later on in the class do seem to still retain a high level of information learned and this retention is reflected in their assessment results.

Although the collaborative process utilized in the CMS Game Design course is tailored to that particular classroom environment, the basic structure that places the impetus of achievement and learning squarely on the shoulders of student-led teams is applicable to a variety of classroom settings. The assessment tools are merely means for the instructor to quantify the learning that occurs when students are allowed to interact in meaningful ways. Creating a classroom based on collaboration and individual responsibility is an effective way to increase student retention and achievement while decreasing instances of copy and paste projects and poor assessment results. As fewer hands are raised for teacher help during a class period and on-task, loud, excited conversation takes over, it becomes apparent that the potential for student learning may not rest only on the shoulders of the teacher. The latent ability of students to learn through communication is one that should be exploited often and in as many classroom scenarios as possible. While standards-based testing may evaluate students' ability to perform and be successful as an individual, the learning that will allow this success is all about collaboration.

ACKNOWLEDGMENTS

Thank you to the Scalable Game Design research team and Kelly Boren who brought Game Design and Computational Thinking to Carson Middle School.

REFERENCES

[1] V. Barr and C. Stephenson, " Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?", ACM Inroads, New York, NY, 2011 March, Vol. 2, No. 1, 48-54.

[2] A. Basawapatna, K. H. Koh, A. Repenning, D. C. Webb, and K. S. Marshall, K.S., "Recognizing Computational Thinking Patterns", SIGCSE'11, March 9-12, 2011, Dallas, Texas.

[3] A. Basawapatna, A. Repenning, and K. H. Koh, "Closing the Cyberlearning Loop: Enabling Teachers to Formatively Assess Student Programming Projects". SIGCSE '15, March 4-7 2015, Kansas City, MO, USA.

[4] CSTA Standards Task Force. (2011). K-12 Computer Science Standards. Retrieved on September 1, 2015 from http://www.csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf.

[5] A. Dahotre, Y. Zhang, and C. Scaffidi, "A Qualitative Study of Animation Programming in the Wild", ESEM'10, September 16-17, 2010, Bolzano-Bozen, Italy.

[6] A. Repenning and A. Ioannidou, "Broadening Participation Through Scalable Game Design", SIGCSE'08, March 12-15, 2008, Portland, OR.

[7] L. Seiter, and B. Foreman, "Modeling the Learning Progressions of Computational Thinking of Primary Grade Students", ICER'13, August 12-14, 2013, San Diego, CA, USA.

[8] L. Werner, J. Denner, and S. Campe, S., "Children Programming Games: A Strategy for Measuring Computational Learning", ACM Transactions on Computing Education, Vol. 14, No. 4, Article 24, December 2014.

[9] J. Wing, "Computational thinking". Communications of the ACM, 49(3), 33–36.