Assignment 3

Lesson Plan: Scratching the Bee-Bot

Marwa Kotb

ETEC530 (66A)

Constructivism Strategies for E-Learning

Introduction

Recently, the British Columbia (BC) curriculum has set the need for a basic understanding of computer science (CS) and computational thinking as an essential goal for students (BC Ministry of Education, 2018). In this assignment, I am presenting a lesson plan titled "Scratching the bee-bot" for grade eight students supporting the required competency development (thinking, communication, personal and social competencies) and covering designated objectives in the "computational thinking module", Applied Design, Skills, and Technologies (ADST) grade 8 content (BC Ministry of Education, 2018).

The lesson design draws upon the available literature on constructivism, primarily from: The essential criteria of constructivism provided in Baviskar, Hartle, and Whitney (2009) [reddish-Pinkish tags], Murphy's (1997) "constructivist checklist" which itemizes crucial constructivist characteristics required in practice [greenish tags], and constructivist approaches applied in Computer Science Education (CSE); they were suggested in (Ben-Ari, 2001) [bluish tags]. All elements are shown in the table below.

Essential criteria of constructivism (Baviskar et al., 2009)	Constructivist checklist (Murphy, 1997)	Constructivist approaches in CSE (Ben-Ari, 2001)
#Activate prior knowledge #Cognitive dissonance #Realistic application with feedback #Reflection	# Authentic context and activities #Authentic assessment #Consideration of error #Scaffolding #Social interaction	#Bricolage programming #Don't run to computers #Don't start with abstraction # Minimalism

Table 1: Constructivism elements from literature

Part A: Course concepts and literature review

This section will shed light on the literature, principles, and practices with which the structure and components of the lesson and assessment plans sit. I will start with the type of knowledge acquired in computer science (CS) and the constructivist epistemology placement in

terms of that. Following that, I will unfold the theoretical perspectives and foundational elements of constructivism that are congruent with CS education and presented in the plans. Lastly, I will discuss constructivist instructional approaches and evaluation methods that were applied in the plans.

1- Constructivist epistemology and computer science

Before planning a constructivist lesson in the computer science (CS) field, one must consider the relationship between the constructivist epistemology and CS. In the article "Constructivism in computer science education," which was recently discussed in the "Coding and constructivism" research café (Fleming, 2020), Ben-Ari (2001) drew attention to an uncontroversial observation that the behavior of computers is by design reliable and predictable and the syntactic and semantics of coding are non-negotiable. He outlined that "the computer forms an accessible ontological reality" (Ben-Ari, 2001, p.56). This comes as no surprise for a discipline that embodies naturalism or the laws at hand (Colburn & Shute, 2010); the successful performance would require arriving at the right solution (Beynon, 2009). And thus, Ben-Ari (2001) concluded that there is no room for the "constructivist epistemology" that describes the knowledge not as truths in the CS discipline. As such, it seems that the nature of CS is consistent with the standard definition of knowledge as necessarily involving truth and is leaning towards realist conception of objective truth (Pritchard, 2018); after all the proper scope of the CS as a discipline is deemed as science (Beynon, 2009).

However, there is a complementary story to be told in this aspect. As CS shares more with engineering than it does really with science, Colburn and Shute (2010) argue that computer scientists "are after knowledge of effective values" (p.128); which means that they are after the knowledge that is adopted because it worked or viable rather than uncovering objective truths as

in the traditional sense of tripartite analysis and realism. This knowledge is in the form of prescriptive laws developed by the coders to help them control the computational operations. Coping with the latter argument that is more endorsed by CS scientists, we are presented with reconciliation with the constructivist epistemology and a broader perspective for the practitioners in the computer science education (CSE). Nevertheless, teaching programming is no longer concerned with getting the answer right only. The recent guidelines in the BC curriculum (2018) emphasize developing students' higher thinking capabilities that enable them to evaluate, explain, and justify the solution process. The drift towards the process and not the product correlates well to Prichard's (2018) in his argument for developing student's intellectual virtues as an ultimate epistemic goal of education stating that: "it is no good being very accomplished in arithmetic (a mere cognitive skill) if one doesn't know how best to employ this cognitive skill to serve one's wider intellectual ends" (p.168-169). It also casts with the constructivist vision, "learning is not the result of development, learning is development" (Fosnot & Perry, 2005, p.55)

2- Constructivist learning theories and computer science

"Constructivism is less a single theory of learning than a collection of perspectives" (Merriam & Bierema, 2014, p.36). They are all based on a fundamental assumption that knowledge cannot be transferred from the external world but instead constructed in the students' minds as they attempt to make sense of their experience (Von Glaserfeld, 2005). Constructivists approve that knowledge is intricate, in the metaphor of Malaguzzi's "a tangle of spaghetti" (Moss, 2004, p.25), which asserts Pritchard (2018) distinction between knowledge and beliefs as the latter are groundless knowledge cases. Consequently, constructivists posit that learning is a dynamic and nonlinear process instead of a fixed commodity (Fosnot & Perry, 2005; So, 2005). They place the students at the center stage of any learning experience and portray the educators

as facilitators; they guide, monitor, assess, affirm acquired knowledge, and challenge the thinking of the students (Fosnot & Perry, 2005). #Activate prior knowledge

Baviskar et al. (2009) presented four essential criteria of the constructivism. Firstly, probing the students' prior knowledge; at the launch of a new topic or experience, it is recommended to ask the students what they know or think that they know, or use simple pre-tests or KWL charts to assess the prior knowledge or skills (Boettcher & Conrad, 2016; Baviskar et al., 2009). Fenwick and Parsons (2009) claimed that the activation of previous experiences supports the accommodation process in the same way one is pulling the appropriate file up on the screen before adding the new information. Boettcher & Conrad (2016) added that by knowing what students know, the educators could design experiences that ensure accurate knowledge models and then support the growth of these models. In the same vein, Hadjerrouit (1999) affirms that the activation process will help reveal students' misconceptions and the programming practices that directly conflict with the new concepts. Moreover, it may support building strong links between the programming language constructs (e.g., the "while" and "for" loops). #Cognitive dissonance

Secondly, an essential notion in a constructivist approach is "cognitive dissonance" (Baviskar et al., 2009, p.544). So (2002) emphasized that "cognitive change only take place when previous conceptions go through a process of disequilibrium with the new information" (Constructivist Theories, Para. 4). Therefore, the educators need to promote that conceptual conflict to help students in building their understanding of reality (Sunal, n.d.); they might use disconfirming ideas, experiments, or demonstrations for that. #Consideration of error

Fosnot and Perry (2005) warned that merely listing or correcting misconceptions to students during the restructuring phase is fruitless; instead, mistakes need to illuminated, explored, and

discussed. Murphy (1997) added that students' errors should be seen in a positive light, as ways to provide feedback. Inline with these viewpoints, Ben-Ari (2001) emphasized that programming misconceptions and poor coding habits should be regarded as a pedagogical equipment rather than as a symptom of failure. #Realistic application

Thirdly, "application of the knowledge with feedback" (Baviskar et al., 2009, p.544). So (2002) highlighted that constructivist learning is more effective using methods that typically involve generative learning, questioning or inquiry strategies, hands-on inquiry, and applications to real-world situations. These methods are consistent with the prospects in the CSE; Hadjerrouit (1999) emphasized the importance of incorporating learning by doing to engage the minds of the students appropriately, so they successfully tackle the challenges of the coding approach. Ben-Ari (2001) also suggested that programming activities should encompass realistic, intrinsically interesting problem situations that mimic the future tasks and problems in the context; the tasks should allow the students to be exposed to social interplays and accustomed to learning resources. #Feedback

According to Hattie and Timperley (2007), feedback has a powerful impact on students' learning and achievement. In their framework, they set up three perspectives of feedback: "feed up" (where am I going?), "feedback" (how am I going?), and "feedforward" (what is next?). They also differentiated between the types of feedback according to the level of cognitive complexity: At the task level, the students receive feedback about the content, or facts (e.g., is the output of the logic operation correct or not?). At the process level, feedback focuses on the strategies of student performance (e.g., What might be needed to master loops?). At the regularity level, feedback aims to promote a higher skill in self-evaluation or confidence to engage further on a job (e.g., What can be done to manage, guide, and monitor the way of

learning?). Last, at the self-level, feedback focuses on the recipient; this level is undesirable and the least sufficient as it doesn't relate to the specific educational goals. #Feedback

Fenwick and Parsons (2009) provided a few practical tips for providing feedback that might be helpful for CS educators in the practical coding context (i.e., lab-work); their tips concur with Hattie and Timperley (2007) discussion. Firstly, “being selective” (Fenwick & Parsons, 2009, p.226), meaning that the feedback should focus on one or two specific things; the students cannot learn everything at a time. Secondly, immediacy is required; the students tend to internalize the feedback that comes directly after a performance. Thirdly, being relevant and precise or otherwise, the input can easily be ignored and be of a low-value to students.

The authors also suggested using instruments such as “observation checklists” and “formative feedback notes” as guides while giving feedback in practical tasks (Fenwick & Parsons, 2009). The templates may consist of simple criteria and comments for what can be done to improve the students’ work (Fenwick & Parsons, 2009). #Reflection

The last criterion in Baviskar et al. (2009) is “reflection”; the authors emphasized that the students need to be aware of the learning they acquired. The most significant challenge when applying this criterion in the coding context is that students focus on the product rather than on the solution process (Hadjerrouit, 2001). For resolution, in the reflective activities, students should be asked questions that probe the concepts underlying the programmable solutions (e.g., what was easy and difficult about coding the Robot and why?). #Social interaction

Ernest (2012) warned that “there is the risk of wasting time by worrying over the minutiae of differences” (p.459) between the varying theories and perspectives of constructivism. Instead the coordination and combination of the various forms will provide “a broad base for interpretation and for practice” (Murphy, 1997, p.14). Casting with the viewpoint, So (2002)

staged learning as personally constructed on the one hand and the other mediated in social contexts and through social negotiation, collaboration, and interaction. An # Scaffolding important concept for social constructivists is scaffolding, which enables students to perform tasks beyond their abilities via the assistance of a More Knowledgeable Other (MKO) (Murphy, 1997). The notion is correlated to the concept of the zone of proximal development (ZPD) that defines the individual's state of readiness for learning; fostering the development within this zone via scaffolding strategies (e.g., visual aids, learning resources, etc.) leads to the most rapid learning (Boettcher & Conrad, 2016). However, Norton and D'Ambrosio (2008) in their teaching experiment observed that student's developmental readiness does not solely depend on his or her ZPD but also on the zone of potential construction (ZPC) which is "a hypothetical reorganization of a students' present ways of operating" (p.236); their results emphasize the argument for rapprochement between the constructivist positions in practice (Cobb, 2005).

Merriam and Bierema (2014) argue that the experiences from # Authentic context and activities which one learns are these situating the whole process of learning and making learning as authentic as possible. They discussed several strategies that CS educators can employ to make learning more authentic such as fieldtrips, roleplay, case studies, simulation games, and introducing projects or problems from outside the classroom. These methods encourage dialogue with the educator and collaboration among students and echo Vygotsky's emphasis on learning as a socio-cultural activity (Fosnot & Perry, 2005).

3- Constructivist instructional approaches for computer science

Programming is often described as the "wicked problem" (Moons & De Backer, 2017, p.97); the subject matter involves abstract and complex approaches such as #Bricolage programming abstraction, recursion, and iteration to process and analyze data, and to create real and virtual

artifacts, which are often hard to teach and learn particularly to youth (Moons & De Backer, 2017). Ben-Ari (2001) suggested several constructivist pedagogies for teaching CS that may be used to approach this problem. One of these strategies is bricolage, commonly coined with "tinkering" programming; the approach draws on the concept of "the science of the concrete" instead of the analytic methodologies in the Western science (Rose, Habgood, & Jay, 2017). The block-style programming that is widely used nowadays in CSE draws on the bricolage theory; they support the novice in building a concrete reification of the computational thinking concepts and facilitate their comprehension of the abstract concepts of coding # Authentic context and activities (Moons & De Backer, 2017). Scratch the computational platform employed in the presented lesson plan is a manifestation of bricolage as it allows a variety of representations for the students to construct their knowledge (e.g., spirits, drawing, sound, motion, sensing, etc.). Moreover, the tool allows learning by doing, for example, creating animated stories, multimedia presentations, games, simulations, and other interactive projects. Furthermore, #Consideration of error it enables "an endless debugging: try it and see what happens" (Ben-Ari, 2001, p.63); thus, the youth would recognize that the failure of the making-activity is a feedback for the path toward mastery rather than becoming helpless, fearful, or overwhelmed. Though, there are potential upsides in the use of Scratch and a confirmation of its match with the constructivist perspective, as discussed in Fleming (2020). However, educators should be mindful that "the tool is merely a means; it is almost never an end" (Van der Meij & Carroll, 1995, p.248), the effective "integration must be supported with educational theory and sound pedagogy" (Winter, 2017, p.178). # Minimalism

Another approach that was suggested in Ben-Ari (2001) was using the "minimalist instruction" technique. The core idea of minimalism is starting the lesson or unit of learning with

a real task or a problem for students to solve and learn through and to reduce the reliance on passively teaching conceptual material (e.g. reading text, listening to lectures, etc.) (Van der Meij & Carroll, 1995).

#Don’t start with abstraction

A third constructivist-based teaching technique from Ben-Ari (2001) is not to start with abstractions; that is, to teach procedural coding first. Ben-Ari (2001) argues that beginning with the object-oriented approach is difficult for novice students as it is more abstract than the procedural approach. He added that from a constructivist perspective, the object-oriented model's proponents don't take into account the students’ prior knowledge and reject Piaget’s view that abstraction follows assimilation; consequently, the students fail to construct a viable model of a computer (Ben-Ari, 2001).

#Don’t run to computers

The last strategy is “don’t run to the computer” (i.e., what is essential is not the language, but how to solve a problem without a code). The approach suggests that students need to refine their understanding of a problem before rushing it up to the computers. Ben-Ari (2001) argues that the procedure is consistent with the constructivist approach and will enable students to construct a good model of the problem or task in hand. In CS practice, an educator might set a mind map to help the students break the problem down into smaller single-concept(s) or pieces or perhaps conduct a discussion on how students can formulate the problem statement or compose their algorithmic solutions. Once the students have a more mature picture and analyzed the problem, they can start translating their thoughts into codes and test their ideas.

4- Constructivist assessment methods for computer science

It is commonly advised to employ Bloom’s Taxonomy when planning assessment; the taxonomy can help educators vary the kind of questions they ask and the type of tasks they use to evaluate the students’ growth (Fenwick & Parsons, 2009). Boettcher and Conrad (2016) framed

the assessment in terms of Bloom's revised taxonomy in one of three levels. The first level "assess facts and concepts," (p.207) aligning with the foundational processes in the taxonomy (i.e., remember and understanding). The second level is "assess with simple doing tasks" (p.207) (e.g., preparing a podcast), aligning with the taxonomy's middle process. And the third level is "assess using complex creating projects" (p.207) (e.g., building a programming game), aligning with the taxonomy's upper process. Despite the assessment level, based on the constructivist view, any assessment would need to support individualized learning and afford students the opportunity to construct and apply knowledge (Boettcher & Conrad, 2016). #Authentic assessment

Projects and hands-on experiences were recommended as ultimate constructivist-inspired methodologies for measuring performance in programming knowledge (Ben-Ari, 2001). They would enable the application and adaptation of knowledge to new situations. Additionally, they would capture the more complex and transversal skills and competencies that are required within the recent shift towards competence-based curricula (Redecker & Johannessen, 2013). For successful implementation of this form of authentic assessment, the design should enact multiplicity and flexibility (i.e., ultimately enable students to set their personal goals) (Boettcher & Conrad, 2016). Also, feedback need to be frequent and ongoing so that students have a reasonably clear idea on how they are doing throughout their applications and projects (Boettcher & Conrad, 2016).

Part B: Lesson Plan

1- The bee-bot mission learning module

The presented lesson is part of a learning unit titled "The bee-bot mission"; the unit is intended to cover three key objectives of the "computational thinking" module:

- "Software programs as specific and sequential instructions with algorithms that can be reliably repeated by others
- Debugging algorithms and programs by breaking problems down into a series of sub-problems
- Programming languages, including visual programming in relation to text-based programming and programming modular components" (BC Ministry of Education, 2018).

Table 2: Curricular objectives

Minimalism # Authentic context &

The whole unit is framed to the students in the form of an assigned mission where the students are tasked to work together in pairs and develop their very own version of the well-known bee-bot game; teams are recommended when working on problem-solving scenarios (Boettcher & Conrad, 2016). The Bee-bot game is based on the original Bee-Bot classroom- programmable floor robot designed to support younger children to improve their algorithmic thinking skills via programming sequences of forwards, backwards, left, and right 90-degree turns.

Authentic context & activities #Social interaction

Throughout the assigned mission, the students will integrate multiples subject skills. They will analyze the problem and break it down into smaller parts, develop their algorithmic solution, implement the game using Scratch, and use a series of test cases to verify that a solution performs according to their design specifications. Besides, students will also strengthen their collaboration and communication skills as they discuss stimulating ideas with teammates and the whole classroom, present information, findings, and supporting evidence concisely and logically (i.e., core competencies).

The module content is split into ten lessons, the allotted time for the session is 80 minutes. The software SDLC (Software Development Life Cycle) is used as a guide to structuring the learning experience and the developing of the bee-bot game; each phase will feed into the next step. Throughout the lessons, the students will commit to the particular computing vocation in the SDLC phase and immediately apply the lesson content in the bee-bot application.

2-Unit plan:

<table>
<tr><td colspan="4">Bee-bot mission: Unit plan
Computational thinking module
Grade level: 8</td></tr>
<tr><td>Number of lessons: 10
Equipment & IDE:
-Laptops or tablets
-bee-bot app
- Scratch 3.0
-4 to 6 Bee-bots (floor robots)</td><td>Assessment:
-of Learning: Bee-bot game in five phases
-for learning: Hands-on practices and check points
-as learning: Reflections and self-assessments</td><td colspan="2">Instructional methods:
#Bricolage programming
#Don’t run to computers
#Don’t start with abstraction
#Minimalism</td></tr>
<tr><td rowspan="2">Lesson title</td><td>Learning objectives</td><td>Learning tasks</td><td rowspan="2">Student role</td></tr>
<tr><td>By the end of the lesson the student will be able to:</td><td>The Students are required to:</td></tr>
<tr><td>The mission</td><td>• Explain the problem context</td><td>• Interact with the physical programmable floor robot bee-bots
• Operate the functionality (control buttons)
• Experiment moving the bee bot with different starting points, different formations for the objects and distances.</td><td>Inquirer/ Explorer</td></tr>
<tr><td>SDLC phases</td><td>• Identify and explain the steps of the Software Development Life Cycle (SDLC)</td><td>• Explore the bee-bot game
• Discuss how to use SDLC to develop their own version</td><td>Inquirer/ Explorer</td></tr>
<tr><td>Analyzing the bee-bot using problem statement [Phase 1]</td><td>• Compose the problem statement for a given programming task</td><td>• Project Phase [1]: Determine the problem</td><td>Analyst</td></tr>
</table>

		statement for the bee-bot game	
Designing the bee-bot algorithm [Phase 2]	• Compose algorithm using basic programming structures (i.e., sequential, conditional, repetition)	• Project Phase [2]: Compose the algorithmic solution for moving the bee-bot	Designer
Designing the bee-bot flowchart [Phase 2]	• Use flowchart to solve a programming problem	• Project Phase [2]: Draw the flow chart for the algorithmic solution	Designer
Programming languages [Phase 3]	• Distinguish between the types of programming languages	• Project Phase [2]: Compose the pseudocode	Coder
Scratching the bee-bot [Phase 3] (Two sessions)	• Identify and use appropriate coding environments for a production • Apply Scratch programming • Create a realistic application.	• Project Phase [3]: Implement one level of bee-bot game	Coder
Testing and debugging the bee-bot [Phase 4]	• Apply the testing and debugging techniques	• Project Phase [4]: Test and debug the bee-bot game	Tester/ Debugger
Presenting and sharing the bee-bot [phase 5]	• Describe the programming process and provide reasons for their solution and modifications • Evaluate the ability to work both as individuals and collaboratively • Reflect on the design thinking and processes and identify new design issues.	• Project Phase [5]: Demonstrate their product	Presenter & Deployer (Audience are customers)

3- Scratching the bee-bot: Detailed plan

The lesson plan follows Karplus's three-phase learning cycle: 1) Exploration 2) Explanation 3) Application (Sunal, n.d.). The model is balanced and suitable for the novice young CS students, as some might struggle to extract all of what they need to know from their

own experiences without any guidance. As per the discussion held in the "Failure of Constructivism" research café (Zeng, 2020, July), educators might need to employ constructivism in moderation and balance with guided instruction (if needed) as well as effective evidence-based learning strategies to ensure that the novice students can pick up the appropriate basics and procedures of the discipline.

Lesson 7: Scratching the bee-bot
Grade level: 8 **Session time:** 80 mins **Student role:** Coder
Overview In this lesson, the students will take the bee bot mission to the implementation phase [3], each group will code their algorithmic solutions that was composed throughout the previous lessons (refer to the unit plan). The selected platform is Scratch as the students completed a whole unit on Scratch "Diving deeper with Scratch" at a previous time in the term.
Learning objectives • Identify and use appropriate coding environments for a production • Apply Scratch programming • Create a realistic application.
Equipment and preparation • Laptop or tablets (at least one device for two students) • Access to internet • Scratch group accounts were set and shared with students
Introduction (10 mins)
#Activate Prior knowledge #Cognitive dissonance #Social interaction #Authentic context #Minimalism #Don't run to computers #Don't start with abstraction **Bridge in [2 mins]:** The educator places the bee-bot on the table or desk in front of the students and starts a discussion about the programming phase of the SDLC by asking relevant questions, like: • How would you implement the algorithmic solution you composed to move the bee bot? [use a coding language] Who are you today? [coder] **Activity #1: Which language? [2-3 mins]** (i.e., activate prior knowledge) Following that, the educator asks each student to suggest a coding language for implementation and specify its type (block or text) in the "collaborative Mind map". **Note:** The map is a shared with the whole class. students don't need to sign up to participate, they can just add their name to indicate their entries. **Discussion [5 mins]:** After the students finish the task, the educator asks the students to briefly justify their choices in the mind map (she may pick up the inapplicable choices). If student[s] chose to implement the application using a text-based language such as python, the teacher would possibly ask provocative questions such as

- Would it be feasible to implement the one-level game in about two hours or less using python? Then she highlights that feasibility is an important aspect that the coder must be alert to when making the selection of the coding language (i.e., conflict).

Phase [1] Explore (15 mins)

#Cognitive dissonance #Scaffolding #Minimalism #Don't start with abstraction

The educator reminds the students that they need to start any phase in their "mission" by gathering data about their tasks and roles. Then she invites all students to explore how scratchers implemented their versions of the bee-bot game and for that they need to complete Activity [1] (i.e., no further guidance is provided):

Activity [1]: Know your task [12-15 mins]

- Visit https://scratch.mit.edu/studios/1525526/.
- Explore one solution of your choice
- Read the instructions on the right of the screen
- Use the "Green Flag" to start the game and the "Red Circle Button" if you want to stop.
- Use the key arrow buttons to move and turn the Bee-Bot and press "GO" to send the bee on its way.
- Click on "See Inside" to view the code
- Record your notes and observation in the "The observer sheet"

While the students are completing their task, the educators need to:

- encourage students to explore fearlessly (she may say "keep up with as many levels as you can")
- allow peers to examine one another observations, ask each other questions, and share what they discover.
- take notes of what knowledge students are discovering and what misconception they have, so she may use the information in the next phase of the lesson (she may use the observation checklist, see the next section)
- question students' observations using prompts (i.e., create conflict) such as:
  - ✓ Can you give me more details why do you think you would need the forever block or if-then block?
  - ✓ The coder in this solution used a sentinel loop, why not a counter-controlled loop?
  - ✓ Is this script written for the sprite or the stage?
  - ✓ What do you think will happen if you incremented the move steps by 10 instead of decrementing it?

For students who are struggling to finish the activity, the teacher might support them, pair them with students who progressed quickly, or refer them to watch a tutorial video or access an online resource (see bee-bot resources below).

Phase [2] Explain (15 mins)

#Feedback (where am I going) #Social interaction #Scaffolding #Don't run to computers

The teacher discusses with students the common observations (she may use the coding tips below to further students' understanding if needed). Then she will discuss with the students the requirements in the "**starter tasks**" for the third phase (i.e., few steps to get started with the coding) which is as follows:

Activity [2]: Starter tasks

- Add/draw the spirits of the choice (e.g., bee and flower)
- Add/draw background of the choice (e.g., grid or maze)

- Record dimensions
- Make the sprite face the direction it is moving
- Hook up the "move" block to the up-down-right-left-right arrow keys (or mouse clicks).

Discussion [5 mins]:
Before heading to the devices, the educator invites all groups to ask clarifying questions related to their customized algorithmic solutions. She may offer guidelines, direct them to useful resources, or get them to brainstorm the coding scenario[s] in a more detailed manner. However, she needs to avoid offering the right solution.

Phase [3] Apply (35 mins)

#Realistic application #Feedback (how am I going & what is next?) #Social interaction #Consideration of error #Authentic context & activities #Bricolage programming
Groups access their accounts and start scratching the "starter tasks" for their algorithmic solutions
The educator needs to:

- circulate around the room and observes each pair of students working together
- provide immediate, precise, and relevant feedback on students' work. Feedback should focus on task and process, so for example, if the student used a sequence of the same blocks, the teacher may tell him "Do we know any control structures that can repeat lines of code instead?"
- use the errors as a mechanism to provide feedback on learners' understanding. For example, if the student failed to move the spirit to left or right along the grid, she might remind the students of the number line and its relationship to left and right movement;
- normalize errors, so she might say that "errors are normal part of the coder life, remember you need to code, test, and debug so you can become a good coder"
- encourage the students to debug their errors on their own, for example, ask them "what have you tried so far? Ok now check scratch Wiki and examples for direction and angles and you will be able to fix this issue".
- offer feedback on how they might improve and elaborate to the next level? For example, for groups who are progressing quickly through the task, she might suggest to add more challenge to the level in their game.
- use the "observation checklist" to assess students' progress and provide feedback tailored to group's or student's needs (see next section).

Closure Activity (5 mins)

#Reflection
Teachers ask students to individually complete the "exit ticket" (i.e., assessment *as* learning)
Activity [3]: Exit ticket
In one to two statements answer the following:

- How is implementation different from the algorithm for your bee-bot game?
- What was easy or difficult about scratching the bee-bot? Why?
- What would you like to create next?

Extension
Students will be given 2 more sessions to complete and debug their projects through a written assessment (see assessment *of* learning). In the final stage, students will present their projects to their peers, reflect on their own design and development process, as well as receive feedback from the teacher and their peers (refer for unit plan).

Bee-bot resources

- Berry, M. (2017). How to make your own Bee-Bot emulator in Scratch [Video Post]. Retrieved from https://www.youtube.com/watch?v=sljU21uNFJQ
- Malcolm,G. (2011). Scratch and Bee-Bots - Lesson 2 - making the bee-bot change colour [Video Post]. Retrieved from https://www.youtube.com/watch?v=wrcVdjLNLIE
- Malcolm,G. (2011). Scratch and Bee-Bots - Lesson 4 - adding a stage background [Video Post]. Retrieved from https://www.youtube.com/watch?v=wrcVdjLNLIE

Coding tips

1. start
2. While (arrow up key is pressed)
3. move 10
4. End while
5. While (arrow down key is pressed)
6. move -10
7. End while
8. While (arrow right key is pressed)
9. turn 90 degrees
10. End while
11. While (arrow left key is pressed)
12. turn -90 degrees
13. End while
14. Stop

4- Assessment and evaluation

For assessment *for* learning, the students are going be assessed using the observation checklist, the instrument will help the educator to record the student's performance in the exploration and application phases of the lesson and offer precise and relevant feedback.

Observation checklist			
Criteria: Not Meeting [0] Approaching [1] Meeting [2] Exceeding [3]			
[1] Exploration phase skills and requirements			
[A] Students was able to reply to the questions in the observation sheet. [B] Students connected and engaged with others while exploring the Scratch solutions			
Student Name	Skills [A/B]	Criteria [0/1/2/3]	Comments
[2] Application phase skills and requirements			
Category [1]: Technical skills Students were able to A. Add/draw the spirits of their choice B. Add/draw background C. Make the sprite face the direction it is moving D. Hook up the "move" block to the up-right-left-right arrow keys. E. Develop a working and creative solution. **Category [2]: Group work** F. The student contributed in the production. G. The student was able clearly communicate workable ideas with his/her teammate.			

Student Name	Skills [A/B/C/D/E/F/G]	Criteria [0/1/2/3]	Comments

For assessment *of* learning, the student will complete their Mission check [3] (project work); they will be graded using a guiding rubric, following the implementation phase, they need to reflect on their work and self-assess themselves using the grading rubric (i.e., assessment *as* learning). #Authentic assessment

Mission check [3]: The bee-bot game using Scratch 20%
Description: As a group, use your creativity and knowledge of programming and Scratch to create a **one level** of the bee-bot game. Feel free to look through the applications that come with Scratch for inspiration, but your work as a team should not be terribly similar to any of them. Try as a group to think of an idea on your own, and then set out to implement it. If along the way, you find it too difficult to implement some features, try not to fret, and alter your design or work around the problem.
Marking instrument: - Use the rubric as a guide to your implementation, it will also help you self-assess your work. - Note: The same rubric will used for your feedback.
Reflection & self-assessment After completing this phase of your mission, respond to the following questions (i.e., individually): 1. What are the important highlights you learned throughout the role of a coder? 2. What were the challenges you overcome? 3. If you were to do this game over, what would you do differently to improve your work? - Provide a self-assessment mark using the rubric? Give a brief rationale for this mark?

Conclusion

In Part A, I simulated the journey in Etec530 and presented an interplay between the constructivist epistemology, theory, pedagogies, and assessment however in light of the CS practice. In Part B, the lesson plan was a manifestation and translation of several perspectives of constructivism theory and practice. It was framed in a realistic intrinsically interesting problem situation to support the students to construct their CS knowledge and develop their intellectual skills. As teaching and learning are complex processes, refinement and adjustment to the proposed plans might be needed to capture the individuality of students and the reality in the classroom.

References

Baviskar 1, S. N., Hartle, R. T., & Whitney, T. (2009). Essential criteria to characterize constructivist teaching: Derived from a review of the literature and applied to five constructivist-teaching method articles. *International Journal of Science Education, 31*(4), 541-550. doi:10.1080/09500690701731121

BC Ministry of Education (2018). BC's new curriculum. Retrieved from https://curriculum.gov.bc.ca/curriculum/adst

Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE Bulletin, 30*(1), 257-261. doi:10.1145/274790.274308

Beynon, M. (2009). Constructivist computer science education reconstructed. *Innovation in Teaching and Learning in Information and Computer Sciences, 8*(2), 73-90. doi:10.11120/ital.2009.08020073

Boettcher, J. V. & Conrad, R.-M. (2016). *The Online Teaching Survival Guide* (2nd ed). San Francisco: Jossey-Bass.

Cobb, (2005). Where is the mind: A coordination between the sociocultural and cognitive constructivist perspectives. In Fosnot, C. T. (Eds). *Constructivism: theory, perspectives, and practice,* (2nd ed., pp.63-92). Columbia University: Teachers College Press.

Colburn, T. & Shute G. (2010). Knowledge, Truth, and Values in Computer Science. In Vallverdú, J., & InfoSci-Books (Eds). *Thinking machines and the philosophy of computer science: Concepts and principles,* (pp.119-131). Hershey PA: Information Science Reference.

Ernest, P. (2012). The one and the many. In L. Steffe & J. Gale (Eds.).*Constructivism in education* (Taylor and Francis ed., pp.459-486). doi:10.4324/9780203052600

Fenwick, T. J., & Parsons, J. (2009). *The art of evaluation: A resource for educators and trainers* (2nd ed.). Toronto: Thompson Educational Pub.

Fleming, A. (2020). Research café: Constructivism and coding. Retrieved from https://aaronpfleming.wixsite.com/researchcafe

Fosnot C.T. & Perry R. S. (2005). Constructivism: A psychological theory of learning. In Fosnot, C. T. (Eds). *Constructivism: theory, perspectives, and practice*, (2nd ed., pp.26-62). Columbia University: Teachers College Press.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *SIGCSE Bulletin, 31*(3), 171-174. doi:10.1145/384267.305910

Hattie, J., & Timperley, H. (2016;2007;). The power of feedback. *Review of Educational Research, 77*(1), 81-112. doi:10.3102/003465430298487

Meij, H. v. d., & Carroll, J. M. (1995). Principles and heuristics for designing minimalist instruction. *Technical Communication*, *42*(2), 243-261.Retrieved from https://www.jstor.org/stable/43087895?seq=1

Merriam, S. B., & Bierema, L. L.(2014). *Adult Learning: Linking Theory and Practice.* San Francisco: Jossey-Bass.

Moons, J., & De Backer, C. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers and Education, 60*(1), 368-384. doi:10.1016/j.compedu.2012.08.009

Moss, P. (2004) Dedicated to Loris Malaguzzi: The town of Reggio Emilia and its Schools. *Refocus Journal,* 22-25. Retrieved from https://www.sightlines-initiative.com/images/Library/reggio/townofrepmoss.pdf

Murphy, E. (1997). Constructivism from philosophy to practice. Retrieved from https://eric.ed.gov/?id=ED444966

Norton, A., & D'Ambrosio, B. S. (2008). ZPC and ZPD: Zones of teaching and learning. *Journal for Research in Mathematics Education, 39*(3), 220-246.

Pritchard, D., & ProQuest (Firm). (2018). *What is this thing called knowledge?* (Fourth;1; ed.). Abingdon, Oxon; New York, NY: Routledge. doi:10.4324/9781351980326

Redecker, C., & Johannessen, Ø. (2013). Changing assessment - towards a new assessment paradigm using ICT. *European Journal of Education, 48*(1), 79-96. doi:10.1111/ejed.12018

Rose, S. P., Habgood, M.P., & Jay, T. (2017). An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of E-Learning, 15*(4), 297. Retrieved from https://eric.ed.gov/?id=EJ1154629

So, W. WM. (2002). Constructivist teaching in primary science. *Asia-Pacific Forum on Science Learning and Teaching, 3*(1), Article1. Retrieved from http://www.ied.edu.hk/apfslt/v3_issue1/sowm/index.htm#contents

Sunal, D. W (n.d.) The learning cycle: A comparison of models of strategies for conceptual reconstruction: A review of the literature. Retrieved from http://web.archive.org/web/20160426173157/http://astlc.ua.edu/ScienceInElem&MiddleSchool/565LearningCycle-ComparingModels.htm

Von Glasersfeld, E. (2005). Introduction: Aspects of constructivism. In Fosnot, C. T. (Eds). *Constructivism: theory, perspectives, and practice*, (2nd ed., pp.20-25). Columbia University: Teachers College Press.

Winter, J. W. (2018). Performance and motivation in a middle school flipped learning course. *Techtrends, 62*(2), 176-183. doi:10.1007/s11528-017-0228-7

Zeng, M. (2020, July). Re: *Research Café:Failure of constructivism* [Discussion post]. The University of British Columbia Canvas. https://canvas.ubc.ca/