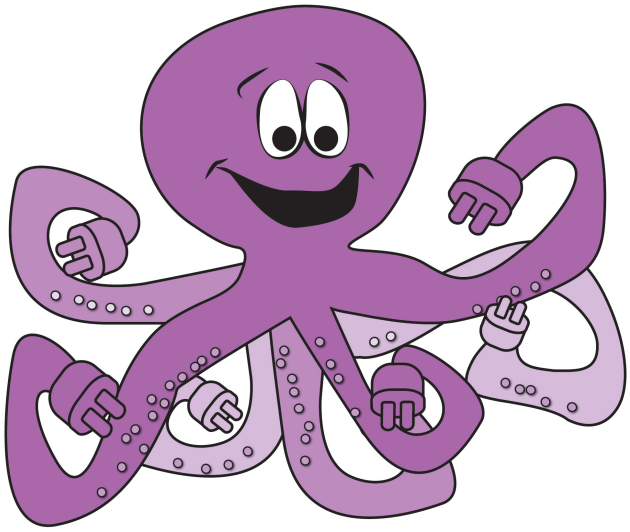
KELP-CS

Curriculum

2015-2016



THE UNIVERSITY OF CHICAGO

CEMSE
CENTER FOR ELEMENTARY MATHEMATICS AND SCIENCE EDUCATION
THE UNIVERSITY OF CHICAGO

Diana Franklin
Computer Science Education
dmfranklin@uchicago.edu

UCSB
UNIVERSITY OF CALIFORNIA
SANTA BARBARA

The Gevirtz School
Graduate School of Education

Danielle Harlow
Science Education
dharlow@education.ucsb.edu

Acknowledgements

Project Directors

Diana Franklin, Center for Elementary Mathematics and Science Education
University of Chicago

Danielle Harlow, Department of Education
UC-Santa Barbara

Development Staff

Hilary Dwyer
Johan Henkens
Charlotte Hill
Ashley Iveland
Alexandria Killian

James Cheng-yuan Hong
Sharon Levy
Timothy Martinez
Iris-Eleni Moridis
Logan Ortega
Kenyon Prater
Jenny So
John Thomason
Rick Waltman

Elementary Teacher Consultants

Larry Kelman
Tracey Schifferns
Janis Spracher

Pilot Test Teachers

Bridget Berg-Gankas
Mary Lou Furrer
Cati Gill
Shauna Hawes
Ashleigh Lemp
Jamie Thompkins
Laurie Thorbjornsen

This project is supported in part by the National Science Foundation Grant #1240985.
Additional support provided by the University of California-Santa Barbara

Table of Contents

Introduction

Welcome to Kids Engaged in Learning Programming (KELP-CS)! KELP-CS is a modular curriculum for 4th-6th graders created by an interdisciplinary research team at UC Santa Barbara. Each of our modules consists of 13-14 hours of instruction, and can be used for each grade level. The first module introduces students to block-based programming and digital storytelling. The second develops these programming skills further and teaches students about game design. Moreover, KELP-CS emphasizes design thinking, the process by which engineers develop innovative solutions to problems. Design thinking is a core part of new science standards for K-12 students, the Next Generation Science Standards (NGSS) and involves understanding the problem, generating ideas, selecting an idea based on multiple constraints, and improving the idea. We see design thinking overlapping with computer science in ways that allow students to access each in new and exciting ways!

Each KELP-CS module consists of activities that are completed either in the classroom or on a computer. During classroom activities, students discuss, collaborate, and engage with computer science without the computer. These off-computer activities introduce and reinforce concepts that students need to apply on the computer. On the computer, students complete small, discrete programming tasks in our interface to learn about a larger computational thinking idea such as sequential motion, event driven programming, and user interaction. As students finish these computer activities, they will transfer these programming skills directly to their design-projects. These design-projects span each module to provide students opportunities to iteratively revise and improve their programs.

For our programming environment, we use a block-based programming environment, called LaPlaya that runs through any Internet browser. The interface is user friendly and age-appropriate upper elementary school students. Instead of programming by typing individual lines of codes, students can snap individual command blocks program. As students progress through our modules, more and more blocks are introduced and at any time they can use our Sandbox area to explore the entire language.

We believe that increasing the opportunities for elementary school children to learn computer science is an essential aspect of preparing students for computer science careers as well as preparing all students to be comfortable and adept with technology. Our goal is to create a curriculum that any elementary school teacher can implement with their class during an academic day. We invite you to explore computer science with us, and help prepare the next generation of computer scientists.

Best wishes

The KELP-CS Design Team

KELP-CS: Scope and Sequence

Module 1: Digital Storytelling

Module 1 is intended as a 4th, 5th, or 6th grader's first introduction to computational thinking and programming. Students learn general programming concepts in the context of a particular language and programming environment (LaPlaya). Some general concepts are the importance of placing things in order in a program, breaking down complex tasks into their basic components, and properly associating code with the events that trigger them, and managing the complexity of several sprites moving at once. LaPlaya programming concepts they will learn are sequential programming, event driven programming, costume changes, and scene changes. As students learn and practice these concepts, they will apply them to their animated story, satisfying design thinking standards.

Module 2: Game Design

Module 2 is geared towards 5th grade students who have completed Module 1 (Digital Storytelling) and are already familiar with basic computational thinking and programming with our block-based, programming environment (LaPlaya). This module teaches students about loops, decision-making, message passing, variables, and the game design process. Further, students will learn more complex programming by implementing these ideas in LaPlaya. Through the module, students will learn about and develop skills using the engineering design process. As a culminating project, they will apply all they have learned to create a game.

Introduction to Computer Science

Overview for Teachers

What is Computer Science?

There is often confusion over the swirl of terms related to computing, technology, and computer science that can be daunting when trying to make decisions about what students need to learn related to computer science. Computer science, as defined by the Computer Science Teachers Association (CSTA) and the Association for Computing Machinery (ACM), is "An academic discipline that encompasses the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society".

Computer science is not just about the use of computers or computer applications, it also includes the knowledge and skills necessary to build the next generations of software and hardware tools that the world needs. The national urgency to improve science, technology, engineering, and mathematics (STEM) education is palpable, as officials have called for reforms in these areas, but what is not clear to policy makers at all levels is that computer science is frequently left out of these initiatives. Computer science drives innovation in all STEM disciplines but it is also a distinct discipline with an extensive body of knowledge.

Computer science teaching sits on a continuum from basic computing concepts that can be attained at elementary and middle school levels to deeper knowledge, skills, and practices more appropriate for secondary school. At the elementary school level students should be introduced to foundational concepts in computer science by integrating basic skills in technology with simple ideas about computational and algorithmic thinking. Learning standards developed by an ACM task force (*A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum committee*) include the following for grades three through five that are included in the KELP-CS curriculum:

- Be comfortable using keyboards and other input and output devices.
- Discuss common uses of technology in daily life and the advantages and disadvantages those uses provide.
- Use technology tools for individual and collaborative writing, communication, and publishing activities to create presentations for audiences inside and outside the classroom.
- Use online resources to participate in collaborative problem-solving activities for the purpose of developing solutions or products for audiences inside and outside the classroom.
- Use technology resources for problem-solving, self-directed learning, and extended learning activities.

Why teach Computer Science?

No other subject will open as many doors in the 21st Century, regardless of a student's ultimate field of study or occupation, as computer science. At a time when computing is driving job growth and new scientific discoveries are happening all the time, gaining a deeper

knowledge of computer science and its fundamental aspects is essential not only to have a clear understanding of "what is going on under the hood" of computer software or hardware, but also to develop critical thinking skills that will serve a student throughout his or her career. The U.S. Bureau of Labor Statistics projects that the computing sector will have 1.5 million job openings over the next 10 years, making this one of the fastest growing economic fields

The knowledge and skills imparted by computer science also enables innovation and opens doors. Many fields of science and business depend on computer science. Despite the incredible diversity of the U.S. workforce, it is clear that most of today's jobs depend on some knowledge of, and skills to use computing technologies. It is also clear that this trend is growing as computing becomes embedded more deeply in everyday commerce and society. Computing touches everyone's daily lives; Securing our cyber-infrastructure, voting in elections, protecting national security, and making our energy infrastructure more efficient are among numerous issues dependent on computing and a strong computing-savvy workforce. If K–12 schools are seeking to make students college- and career-ready, computer science should be part of the core curriculum.

Increasing the opportunities for elementary school children (especially girls and other underrepresented minority groups) to learn computer science is an essential aspect of preparing students for computer science careers as well as preparing all students to be comfortable and adept with technology. Research has shown that students' career aspirations at eighth grade are strong predictors of whether they will attend college and whether they will pursue careers in science or engineering, highlighting the importance of experiences children have prior to eighth grade. Fortunately, more children are gaining experiences in computer science at younger ages. Programming environments designed for children and novices (e.g., Scratch, Alice) are easily accessible by classroom teachers and coding in K-12 education has become a huge movement.

What is unclear to many educators is what curriculum will support this growing trend. Although we are beginning to understand how best to teach computer science at the high school level and middle school level, we know comparatively little about effective instruction at the K-5 level. Luckily, some systematic curricula, such as KELP-CS, have been developed that are targeted at helping elementary school students learn programming and computer science. This kind of curriculum is based upon the higher tiers of Bloom's cognitive taxonomy (involving design, creativity, problem solving, analyzing possible solutions to a problem, collaboration, and presenting work) and develops and extends logical thinking and problem-solving skills that can be applied to real world problems.

How will your students be learning Computer Science?

In addition to engaging in the computation thinking and problem-solving aspects of computer science, students will also be introduced to and gain a foundation in an important aspect of computer science; **programming**. Using the modified Scratch environment, called LaPlaya, to either debug existing programs or create new programs students will learn important computational thinking and programming skills including:

- Sequencing
- Breaking down actions

- Event driven programming
- Initialization
- Animation
- Scene changes

Using the KELP-CS curriculum students will also be engaging in many computer science practices such as precision, optimization, utilizing tools strategically, and switching back and forth between thinking as a software developer and thinking like a software user.

References

Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough: the educational theory and research foundation of the exploring computer science professional development model. In *Proceedings of the 45th Technical Symposium on Computer Science Education (SIGCSE '14)*. Atlanta, GA: ACM.

Liberman, N., Kolikant, Y., & Beeri, C. (2012). "Regressed experts" as a new state in teachers' professional development: lessons from computer science teachers' adjustments to substantial changes in the curriculum. *Computer Science Education*, *22*(3), 257-283. doi:10.1080/08993408.2012.721663

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cuniff, D., ... & Verno, A. (2011). CSTA K-12 Computer Science Standards. *CSTA Standards Task Force*.

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). Running on empty: The failure to teach K-12 computer science in the digital age. Association for Computing Machinery. *Computer Science Teachers Association.*

Zendler, A., & Hubwieser, P. (2013). The influence of (research-based) teacher training programs on evaluations of central computer science concepts. *Teaching & Teacher Education*, *34*130-142. doi:10.1016/j.tate.2013.03.005

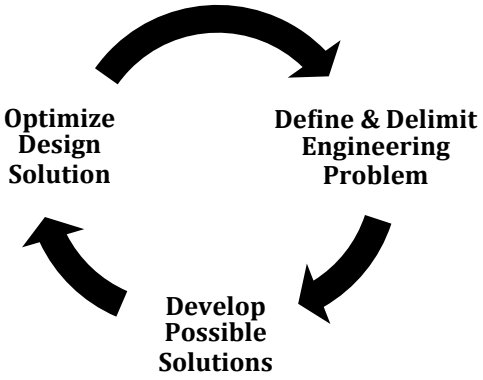
Introduction to Engineering Design Thinking

Overview for Teachers

What is Engineering Design Thinking?

Design thinking is the process by which engineers develop innovative solutions to problems and is now a core idea in new science standards for K-12 students, the Next Generation Science Standards (NGSS). The process involves understanding the problem, generating ideas, selecting an idea based on multiple constraints, and improving the idea. Even before children enter school, they use the process of design thinking to help them solve problems.

Consider a child who lets go of the string of a helium-filled balloon in her kitchen. The balloon, now on the kitchen ceiling becomes too high for her to reach, presenting a problem to solve – How to reach the balloon? She may consider the many resources available to her right in her kitchen such as chairs and cooking utensils. First, she might try using a chair to stand on. If the chair is not high enough, she might try to hold a wooden spoon in her hand to extend her reach while standing on the chair. If she then notices that she can hit the string, but not grasp it with the spoon, she stands on the chair holding a set of tongs, a solution that would allow her to reach and grasp the string. We encounter these sorts of engineering problems and engage in this sort of problem solving or design thinking every day.



The Next Generation Science Standards (NGSS) breaks design thinking into three stages: 1) Defining and delimiting an Engineering Problem, 2) Developing Possible Solutions, and 3) Optimizing the Design Solution The table below describes what students in grades 3-5 should be able to do to demonstrate understanding of these three stages.

Define/Delimit Problem.	Define a simple design problem reflecting a need or a want that includes specified criteria for success and constraints on materials, time, or cost.
Develop solutions.	Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.
Optimize design solution	Plan and carry out fair tests in which variables are controlled and failure points are considered to identify aspects of a model or prototype that can be improved.

The first step is to **define the problem**. An engineering problem includes **goals** (or **criteria for success)** for what should be done (e.g., build a bridge that spans a stream that is 5 meters wide and can support the weight of 5 adults) and **constraints** (e.g., a specified budget, limit on materials or time limits). The first step to solving any engineering problem is to fully understand the problem including the goals and constraints.

The second step is to **develop solutions**. This step includes brainstorming or generating several different ideas and then considering how well each idea is likely to meet the problem goals and staying within constraints.

The third step is to **optimize the solution.** Once an engineer has selected a solution, the next step is to optimize that solution. Optimize means to take an idea and make that idea as good as possible. In engineering, there is usually not a perfect solution and multiple factors are involved. Sometimes as one tries to increase optimization along one factor (e.g., using less material), it may mean decreasing performance along another (e.g., speed).

How is Engineering Design Thinking related to Computer Science & Programming?

Engineering Design Thinking is an important part of computer science and programming. When programmers create a game or an app or any other piece of software, they engage in design thinking. They consider the problem they are solving (e.g., how can I make a fun game for 4th graders to learn about math?) and then develop and optimize their solutions.

How will your students be engaging in Engineering Design Thinking?

In this design thinking activities that complement the programming activities in KELP-CS, children will be developing their own piece of software (a "Digital Story" in Module 1). They will learn tools that are useful to computer scientists like storyboards and flow charts as they develop and optimize their digital stories.

References

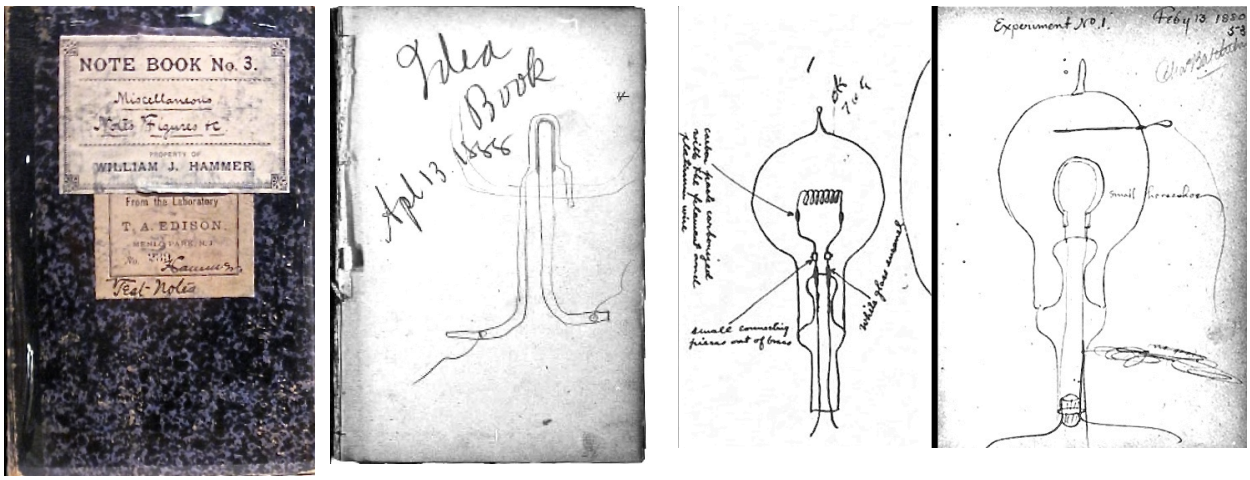
National Research Council. (2012). *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. Washington, DC: The National Academies Press.

Achieve (2012). *Next Generation Science Standards.* Available at http://www.nextgenscience.org/

Using Design Notebooks

Why use a Design Notebook?

The idea of using a notebook to organize thoughts and ideas and keep track of work is not a new idea. Inspiring people throughout history have used notebooks like these to jot down ideas for books or movies (like Mark Twain and George Lucas), record observations of nature (as Thomas Jefferson and Charles Darwin did), keep track of business (like John D. Rockefeller), or flesh out scientific and engineering ideas and designs (as Isaac Newton and Thomas Edison did). Creating a design notebook is now a common practice in engineering, the sciences, and product design. This is a place where individuals or collectives can throw ideas out, incubate them, come back to them, and most importantly, develop them to fruition.



Thomas Edison's Design Notebook with initial sketches of the incandescent bulb.
Photos courtesy of Edison.rutgets.edu

How will your students be using a Design Notebook?

If you chose to have your students utilize a design notebook throughout the KELP-CS curriculum, students will be using either a composition notebook or spiral bound notebook to keep all of their worksheets, notes, and designs together for the "Design Thinking" project that they will be working on throughout each module.

This design notebook should include a title page with information such as student login information (usernames/passwords) and may include a table of contents (if you are planning on going through their design notebooks this is a good way to navigate them more easily).

Throughout the design thinking lessons students will have worksheets that they can cut and paste into the pages of their design notebooks and can then take notes on the subsequent pages, ensuring that everything is in order and easy to find. Once students begin programming using LaPlaya they will be keeping notes on what they did on the computer,

what did/didn't work, and what they changed in their program. They will also be creating basic flowcharts for their program (an essential software development skill and a required course in undergraduate computer science) and will be adding to them as they learn new, complex commands and adapt their digital stories.

A design notebook is ideal for the KELP-CS curriculum because students will be going back and forth between online and offline activities and will not always be working on their Design Thinking project (especially when you consider that they may not even work on this curriculum each day) so by having everything in one place that is easily accessible and in order students can more easily get back into the proper mindset to work on their project and can easily reference previous material if they desire.

The scope of how your students use design notebooks and engage in the design thinking activities is entirely up to the individual teacher and depends on the individual class. You may consider expanding on some of these lessons to incorporate more writing on or off the computer that would allow you to cover multiple standards at once (Common Core Writing as well as Next Generation Science Standards Engineering Design Thinking). You may also want to include more collaboration in these lessons and have students share their work with each other and can write feedback and suggestions in the margins or on the back of the page. This may also be a valuable tool for you to provide feedback to your students or should you choose to evaluate their work.

References

McKay, B., & McKay, K. (2014, January 1). The Pocket Notebooks of 20 Famous Men. Retrieved July 30, 2014.

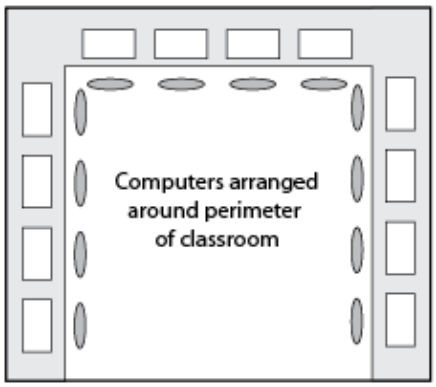
Tips for Teachers

With the help of our pilot teachers and teacher consultants we have compiled a list of tips to assist with common issues that take place in the context of teaching a computer science curriculum in the elementary school classroom such as KELP-CS. We discuss common issues that arise based on the configuration of the learning environment, the mindset of teachers and their students, the grouping of students, and the logistics of working on the computer as well as tips that may help with these issues should they arise in your classroom.

Classroom/Computer Lab Configuration

There are some basic configurations that many classrooms and computer labs fall into and with each configuration comes certain challenges to teaching. Below we will go through some of the possible configurations and common issues that teachers may run into while teaching the KELP-CS curriculum in each.

Around the Perimeter:



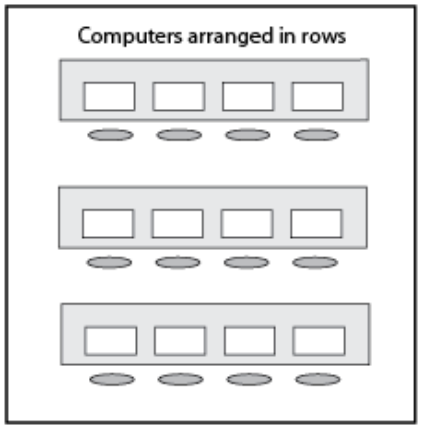
Potential Problems-

When all students face the walls when they are working on their computer it may be difficult for the teacher to get their complete attention when giving instructions are the center of the classroom.

Possible Solutions-

You might consider an easy solution; have the students turn their chairs all the way around to face the center of the room or have the students get up and sit on the floor at the center of the room. Another idea to get students to stop working on the computer and get their attention is to have them put their hands in the air (a big computer stretch) and then turn to the center.

Rows:



Potential Problems-

Because the teacher is usually at the front of the room and cannot see the students computer screens it is difficult to ensure that students are paying attention to the teacher and not distracted with their computer.

Possible Solutions-

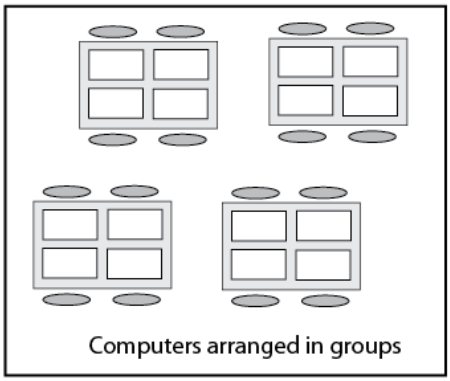
There are several potential solutions for this configuration including having the teacher give directions at the back of the room and having students turn around (like in the "around the perimeter" configuration). You can also ensure that students' screens don't distract them by

having them turn the monitor sideways (if possible), having them turn the monitor off (which shouldn't affect their work only turns off the display), or use a simple piece of paper (shown below) to cover the screen when needed and can be flipped up when students are working on the computer. A piece of paper taped to the top of the computer screen is a great way to ensure that when you're talking students cannot look at their computers. And you can easily see if any student doesn't have their screen covered because it flips to the back where you can see it from the front of the room.

Groups:

Potential Problems-

Like in the "rows" configuration, it is difficult for the teacher to see all students' computer screens at one time and students may be distracted when they should be listen to directions. Another potential problem that may arise is that students will work together more frequently (which is a good thing), but some students may feel less comfortable working with some other students and may take a back seat and let another students do all the work.



Computers arranged in groups

Possible Solutions-

If using personal computers, only allow students to have their computers on their desk when they will be using them. You may even ask students to shut their computers or put them away all together if the class is having difficulty paying attention to instructions. To help with group work you may want to circulate around the room while students are working on the computer and make sure no students is being left out. You may also consider setting some guidelines for students ensuring that all students work on their own computer and must complete their own work unless explicitly stated they work in groups.

Individual Desks:

Potential Problems-

Sometimes students get antsy and move around a lot in their desk. If they are using a personal computer at their individual desk and it is not stable there is a chance that the computer could fall off their desk (which may damage or break it).

Possible Solutions-

You can just warn students about moving their desk too much because the computer might fall or you may consider using a different classroom configuration with multiple desks put together, which would stabilize them and prevent the desks from moving around much.

"Fixed" vs. "Growth" Mindset

Research has shown that there is a continuum of beliefs about where success comes from. The ends of this continuum are called "Fixed Mindset" and "Growth Mindset."
Students with a fixed mindset believe that success comes from innate abilities, while those with a growth mindset believe that success comes from hard work and training. Students with a fixed mindset are likely to fear failure and are less likely to try things that they do not already know they will be successful with. In contrast, students with a growth mindset are likely to continue to work at tasks that they find challenging.

Fixed Mindset	**Growth Mindset**
• Intelligence is innate and does not change	• Intelligence can be developed
• Desires to look smart	• Desires to learn more
• Avoids challenges	• Embraces challenges
• Gives up easily when faced with obstacles	• Persists when faced with obstacles
• Worse performance	• Better performance

Students are likely to come into computer science activities with a wide range of experiences. Students with fixed mindsets may see other students (who have more experience) being successful and determine that they are "not good at computer programming." It is important to encourage a **growth mindset** and remind students that, through experience, they will get better.

Encouraging your students to have a growth mindset
Praise effort, "I see you worked really hard on that task"
Help students see that there are things they can learn even when they are unsuccessful.

For more information, watch the video
https://www.youtube.com/watch?v=xs9fddMg71o

Reference: Dweck, C., (2006). Mindset: The new psychology of success. Random House: New York.

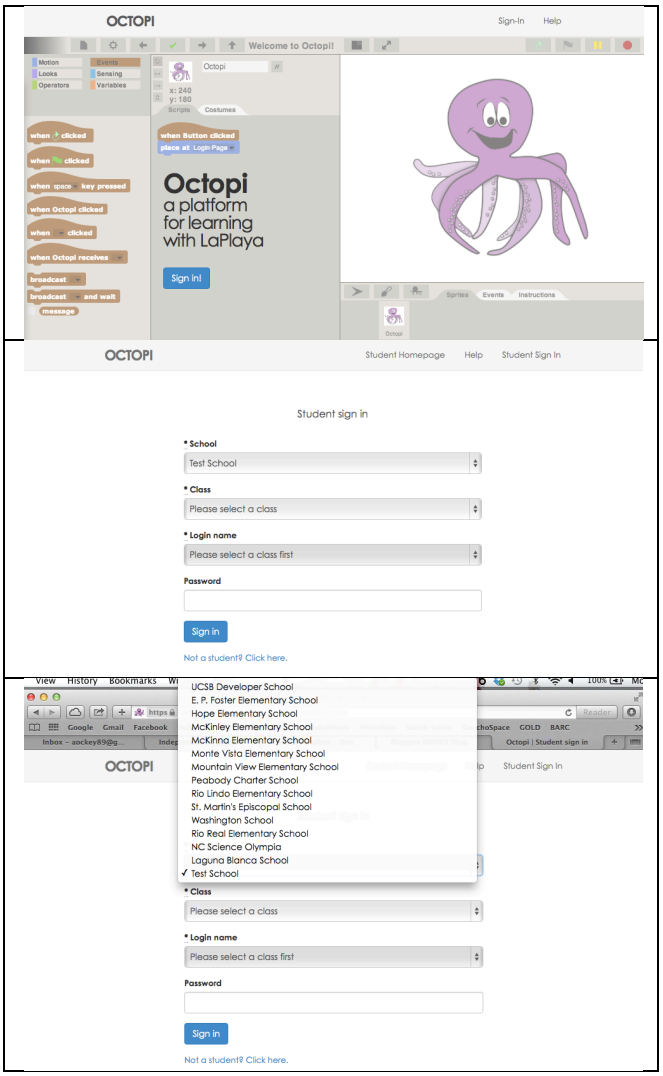
Computer Logistics

Navigating the Internet

Students often struggle with typing in web addresses because of their limited typing skills and their lack of attention to detail. Many students would add spaces or extra punctuation to a web address, which resulted in an error message and much frustration and chaos in the classroom.

To help remedy this situation you can write the web address on the board in LARGE, clear print (You can even write it in all CAPS because you will navigate to the same page regardless of if the address is upper- or lower-case) and emphasize that there are NO spaces in web addresses.

This is also a good time to implement a system for what students should do if they have questions (if you have not already established this in your classroom) because this process usually results in multiple students having difficulty and since there is usually only one teacher students will have to wait before continuing on with the lesson.



Logging In to LaPlaya

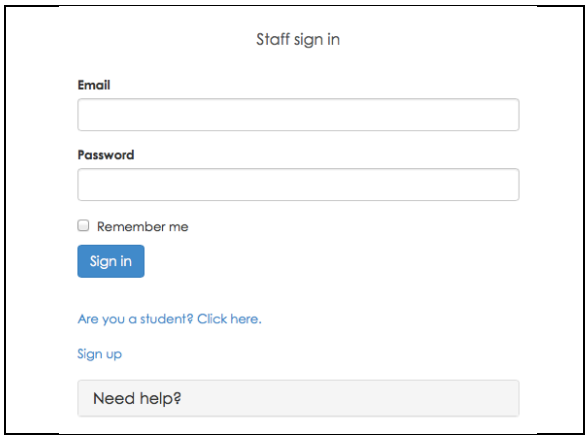
Student Login-
Students should save their login username and password (you can have them write it inside their design notebook or somewhere else easily accessible). You may also want the students to bookmark the web address in their browser (Safari, Firefox, Google Chrome, etc.) so they can easily come back to the website without having to type the address in each time.

Once you get to the website you will see the LaPlaya interface in the background (top picture). Students will click on the blue "Sign In" button to sign in to their account. This will take you to a sign in page (middle picture)where students will select their school from the dropbown menu (shown on the bottom picture) and then select their class and login name (which will be assigned to them).

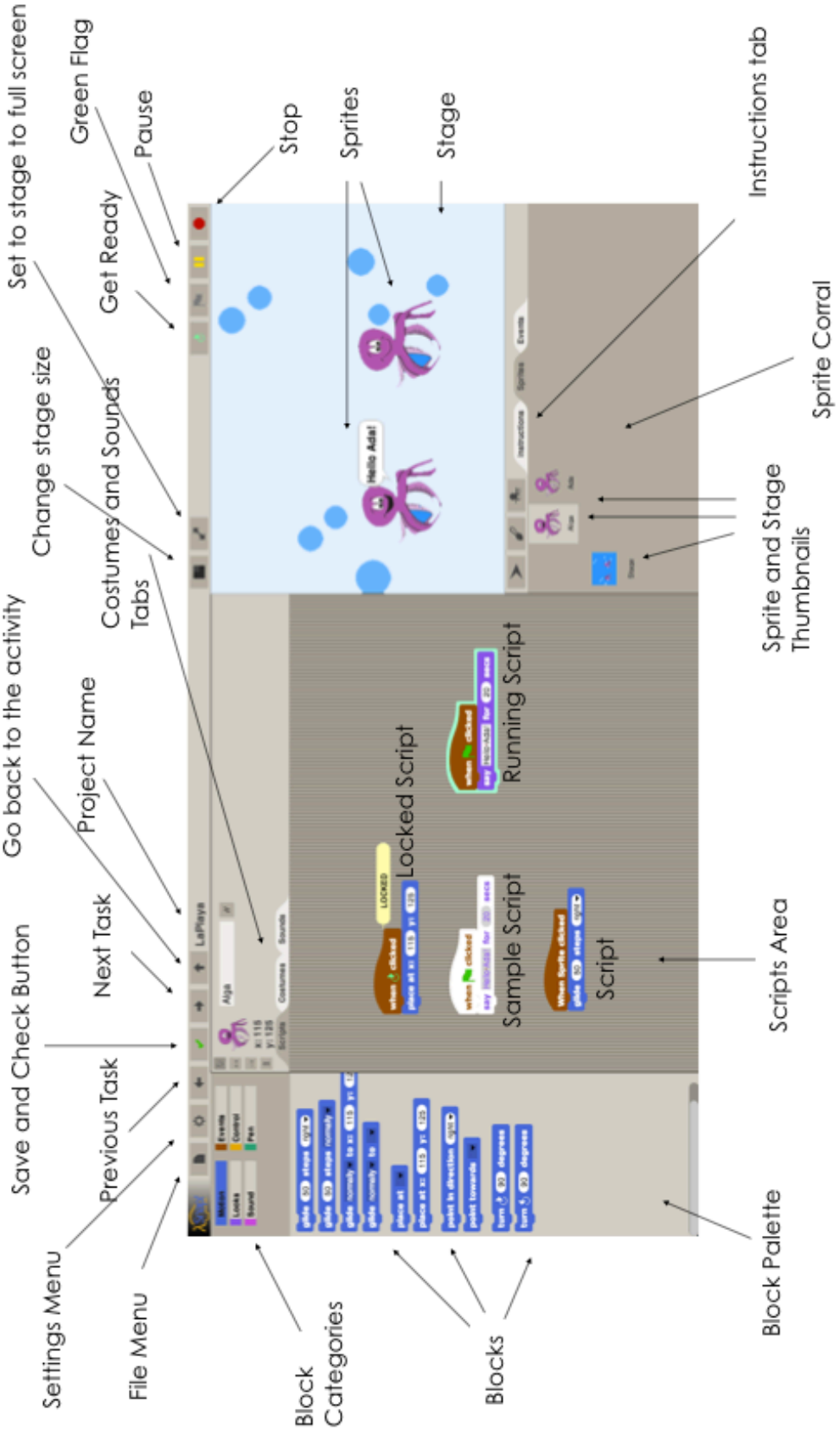
Students will also be given a password that they will need to type in. Their password should not be too difficult, but this is a good time to begin talking about how computer need you to give precise instructions and that it is important to type exactly what they are supposed to or the computer will not recognize it.

Teacher Login-

To sign in to a teacher account you will go to the same web address and click on the blue "sign in" button, but once you reach the log in page you will click on the blue writing at the bottom of the page that says, "Not a student? Click here." This will redirect you to the staff sign in page (shown to the right), where you will enter you email and password. Once signed in you have access to the curriculum, LaPlaya, and information about your class(es).



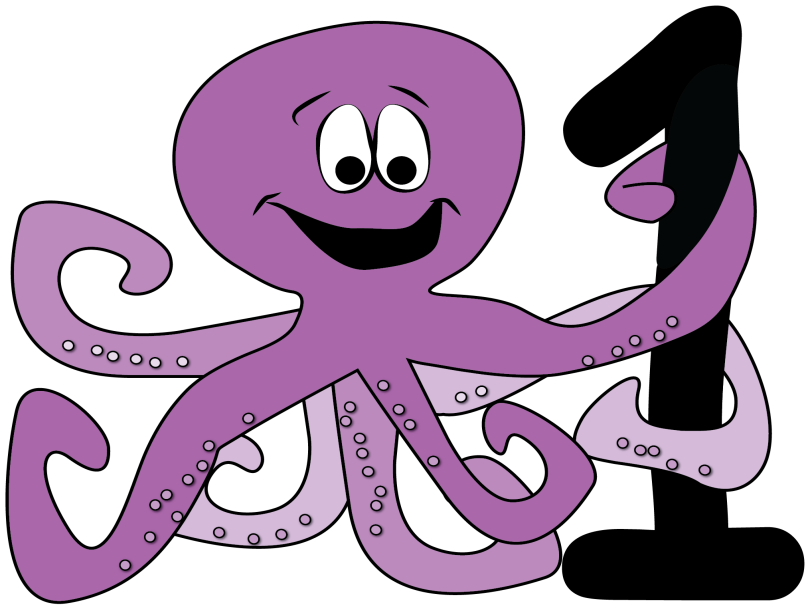
LaPlaya Interface Guide



Module 1

Digital Storytelling

2015-2016



THE UNIVERSITY OF CHICAGO

CEMSE
CENTER FOR ELEMENTARY MATHEMATICS AND SCIENCE EDUCATION
THE UNIVERSITY OF CHICAGO

Diana Franklin
Computer Science Education
dmfranklin@uchicago.edu

NSF

UCSB
UNIVERSITY OF CALIFORNIA
SANTA BARBARA

The Gevirtz School
Graduate School of Education

Danielle Harlow
Science Education
dharlow@education.ucsb.edu

Table of Contents

Module 1 Digital Storytelling

Purpose

Module 1 introduces upper elementary school students to computational thinking and programming with our block-based, programming environment (LaPlaya). For computational thinking, this module teaches students about the importance of order in a program, breaking down complex tasks into their basic components, properly associating code with triggering events, and managing the complexity of multiple objects acting at once. Further, students will learn programming concepts by implementing these ideas in LaPlaya. These include sequential programming, event driven programming, costume changes, and scene changes. Through the module, students will learn about and develop skills using the engineering design process. As a culminating project, they will apply all three areas to create an animated story.

Module 1 takes approximately 13 – 14 hours to complete. Some of the activities are conducted in the classroom (Fired Up); students complete the remaining activities on computers (Wired Up). There are also Design Thinking lessons throughout where students engage in the engineering design process in the classroom to create and complete a digital story by the end of the module.

Overview of Lessons

Day	Activity	Location (Time)	Description
1	Computer Science and Programming	Classroom 45 min.	Students watch a video introducing them to computer programming (Google's *Made with Code*) and discuss how computer science relates to their everyday lives. Following, students explore challenges to receiving and giving directions through a set of FiredUp activities (*Following Directions* and *Drawing Shapes*). • Includes student worksheets.
2	Sequencing	Lab 45 min.	Students create their first programs in LaPlaya and are introduced to sequential thinking in a series of connect the dots tasks.
3	Introducing Engineering Design Thinking	Classroom 15 min.	Teachers lead a classroom discussion: What is design thinking? What is the process? • Includes student worksheets.
4	Breaking Down Actions	Lab 45 min.	Students break down motion into smaller steps by identifying both direction and type of movement. They program a bear sprite to move in different directions to reach a

			honey pot sprite through several scenarios.
5	Defining an Engineering Problem	Classroom 45 min.	Teachers discuss how engineers identify problems and constraints, then introduce students to their design thinking project (the engineering problem and constraints) and students write a problem statement that will guide their work throughout the module. • Includes student worksheets.
6	Event-Driven Programming – When Sprite Clicked	Lab 45 min.	Students begin to engage users in their programs by creating scripts that run on an event: when a user clicks on a sprite. Students program fruit to grow and fall off of a tree (*Fruit Trees, Parst 1 & 2*), and planets to say their names when clicked (*Planets*). As a bonus, students add the notes for each key of a virtual piano (*Bonus: Piano, Part 1*).
7	Brainstorming Digital Stories	Classroom 45 min.	Teachers lead a classroom discussion on the characteristics of creating a good story. Students brainstorm ideas for digital stories that meet the criteria given for their design thinking project. Later, students will write a persuasive piece about why their idea is a good one. • Includes student worksheets.
8	Storyboarding Digital Stories	Classroom 45 min.	Students will learn about the basics of sketching and how storyboards are used to outline the main parts of a story. Later, students will create simple storyboards for their final digital story idea. • Includes student worksheets.
9	Event-Driven Programming – When Key Pressed	Lab 45 min.	Students expand their expertise in event-driven programming by assigning actions to different keys. Tasks include programming a flying rocket (*Rocket* and *Rocket Countdown*), a car cruising California (*California Geography),* and a basket for collecting apples (*Fruit Trees, Part 3*). As a bonus, students can program a digital piano (*Bonus: Piano, Part 2*).
10	Thinking about the User	Lab 45 min.	Students learn the difference between being the user of software and being the designer of software. In these tasks,

			students analyze finished LaPlaya lessons to determine what actions may be hidden and discuss how they can engage the user in their own programs.
11	Flow Charts & The Application Cycle	Classroom 45 min.	Students create flow charts for their digital stories, and plan how the story will move from scene to scene. Students learn about the application cycle of the Engineering Design Process (programming, testing and evaluating, and redesigning) and will be given instructions on how and when to work on their digital story using LaPlaya. • Includes student worksheets.
12	Event-Driven Programming – On Other Sprite Clicked	Lab 45 min.	Students learn how to program sprites when a user interacts with a different sprite. Students program cars to drive when a user clicks a traffic light (*Intersections, Part 1 & 2*), candy to fall when the user clicks a piñata (*Piñata, Part 1*), and fruit to fall when a user clicks a tree (*Fruit Trees, Part 4*).
13	Initializing Programs	Lab 60 min.	Students see the importance of initialization through a demo of two stories, which were initialized differently (*The Tortoise and the Hare parts 1 & 2*). Afterwards, they initialize candy to start inside a piñata sprite (*Piñata, Part 2*) and animals to start behind the starting line in a series of races (*Animal Sprint: Horse, Cat* and *Rooster*).
14	Animating Sprites	Lab 45 min.	Students learn how to use costume changes to create animations. Students create increasingly complex dances for different characters (*Ballerina, Pick A Dancer, Dance Party!)* and then draw and animate their own dancer (*Draw Your Own*).
15	Changing Scenes	Lab 45 min.	Students use a combination of background and costume changes to create multiple scenes. They practice changing scenes to show the life cycle of a plant.
16	Creating Digital Stories	Lab 45 min.	Students are given dedicated class rime to create their digital stories using their programming knowledge and the framework they created in the previous design thinking lessons
17	Finishing Digital Stories	Lab 45 min.	Students are given dedicated class rime to complete their digital stories.

18	Sharing Your Work	Lab 45 min.	Students share their digital stories with their classmates or others. • Includes student worksheets.

1: Computer Science & Programming

Teacher Lesson Plan

Lesson Rationale-

This brief lesson introduces students to computer science and computer programming. Both ideas are connected to the world around students and what they will be doing in this module.

Objectives-

Students will able to:

- Articulate what computer science is and what computer programming is.
- See programming and code in the world around them.
- Relate computer programming to the LaPlaya interface.

Standards Emphasized-

CSTA Computer Science Standards:
L1:6:CT- The student will be able to Demonstrate how a string of bits (code) can be used to represent alphanumeric information & Understand the connections between computer science and other fields.
L1:6:CPP- Identify a wide range of jobs that require knowledge or use of computing.
L16:CD- Understand the pervasiveness of computers and computing in daily life (e.g., voice mail, downloading videos and audio files, microwave ovens, thermostats, wireless Internet, mobile computing devices, GPS systems).

Materials-

Teacher	Student
Device to show a YouTube video with (TV, Projector, Computer, etc.)	Student worksheet titled "Computer Science & Programming" Pencil or pen Design Notebook (optional)

Learning Tasks

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Introduce main concepts to the student: Computer Science, Code, and Programming.	Students use the terminology of the field and vocabulary used throughout the module. *You may want to spend extra time reviewing these concepts to ensure that students understand them fully.*

3-5 min	Watch the Google film "Made w/Code" by going to www.madewithcode.com and clicking "WATCH THE FILM" *(note that this is a YouTube video and some computers may have restricted access to this site)*.	This video introduces how code is small bits of information embedded in the world around us (DNA, words, numbers, etc.). It also reinforces the idea that coding and computer programming are valuable careers, and especially important for girls (who are vastly underrepresented in this field).
5-10 min.	Class discussion about the video and how the concepts they learned relate to one another.	Emphasize the importance of computer science (there will be 1.5 million new jobs in the field by 2018, it teaches important problem-solving skills, and can be used in many other fields) and how code (which is used to program computers) is all around us, and the blocks in LaPlaya are also code.
3-5 min.	Students will answer the questions posed to them on their worksheets after the discussion. These questions help them relate the concepts and tie them to everyday life and the LaPlaya interface they will be using.	Give students a few minutes to respond to their worksheet questions about these concepts. You may want to walk around and help students who may be struggling to find the connections between concepts or relating them to everyday life or the LaPlaya interface.

Name/Number: _____

Computer Science & Programming

What is **computer science?**	The study of COMPUTERS and their PROCESSES including hardware, software, their applications, and their impact on society.
What is **code?**	Small bits of INFORMATION (numbers, letters, words, symbols) that come together to create INSTRUCTIONS for something.
What is **programming?**	Writing the INSTRUCTIONS a COMPUTER must follow (a PROGRAM). This also includes thinking about how a USER may interact with that PROGRAM.

Your teacher will show you a video about **code** and how it's all around us and discuss the importance of **computer science** and **programming**.

Think about the questions below and respond:

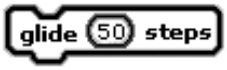
What is an example of **code** that you see in your everyday life?

_____

_____

_____

Are these blocks examples of code? Why or why not?



_____

_____

_____

_____

How does **code** relate to computer **programming?**

_____

_____

_____

Why do you think it is important to learn about **computer science?**

_____

_____

_____

Following Directions Fired Up

Teacher Lesson Plan

Lesson Rationale-

Computers follow a set of directions exactly in the form they are given. Unlike humans, computers are not able to interpret what one says. The ability to communicate a set of directions to other humans in a precise way is a precursor to being able to communicate a set of directions to a computer. The purpose of this exercise is to realize the challenge in such limitations. The end goal is students recognize how precision is needed to make computers do what they want it to.

The result of the lesson will be red and green "tents" for use in determining which students need help in the computer lab.

Objectives-

Students will be able to:

- Follow written directions
- Recognize the precision necessary for written directions to be successful.

Standards Emphasized-

Materials-

Teacher	Student
Directions for students to follow (read aloud to them)	One 8x10 thick red paper for ½ of the students One 8x10 thick green paper for ½ of the students Scissors (optional)

Learning Tasks-

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks
5 min.	Pass out red paper to half of the students and green paper to the other half of the students.
10 min.	Verbally give them directions to make the "tent". Sample directions are given below. You may choose to make these directions more or less vague depending on your particular class.
5-10 min.	Explain what the resulting tents are for in the lab. Discuss what would have made the directions easier. (showing rather than saying directions, knowing ahead of time what the class was supposed to be making, etc.)

Sample Directions:

1) Place the piece of paper on your desk so that it is wide, not tall.

2) Take the right-hand edge of the paper and bring it over to meet the left-hand edge, flattening it to create a fold in the middle of the paper.

3) Unfold it and fold it backwards along the same line.

4) Rip (or cut) the paper along that line.

5) Find someone with a different color piece of paper than you have and trade one of your halves for one of their halves.

6) For each of your two separate pieces of paper (that are different colors):

    a) Place it on your desk so that it is wide, not tall.

    b) Take the right-hand edge of the paper and bring it over to meet the left-hand edge, flattening it to create a fold in the middle of the paper.

Now you have two pieces of paper that will stand up on a desk. If you are doing fine, working on your project, then place the green paper over the red paper. If you need help, place the red one over the green one to signal that you need help. Do not stop working – keep trying to figure out your problem. I will come and help you when I can.

Drawing Fired Up

Teacher Lesson Plan

Lesson Rationale-

Computers follow a set of directions exactly in the form they are given. Unlike humans, computers are not able to interpret what one says. The ability to communicate a set of directions to other humans in a precise way is a precursor to being able to communicate a set of directions to a computer. The purpose of this exercise is to strengthen students' skills in giving directions to other people, including the specific skills listed in the objectives below. The end goal is for students to recognize how precision is needed to make computers do what they want them to.

Objectives-

Students will be able to:

- Write procedures.
- Develop the ability to sequence information logically.
- Incorporate the following language into a procedural text:
  - Appropriate terminology
  - Time linking words
  - Action verbs
  - Precise use of adverbs and adjectives
  - Simple present tense
  - Reference to the reader in a general way (or not at all)
  - Appropriate headings.

Standards Emphasized-

Common Core State Standards

CCSS.ELA-Literacy.RI.4.3 - Explain events, procedures, ideas, or concepts in a historical, scientific, or technical text, including what happened and why, based on specific information in the text.
CCSS.ELA-Literacy.W.4.2 - Write informative/explanatory texts to examine a topic and convey ideas and information clearly.
CCSS.ELA-Literacy.W.5.2 - Write informative/explanatory texts to examine a topic and convey ideas and information clearly.
CCSS.ELA-Literacy.SL.4.4 - Report on a topic or text, tell a story, or recount an experience in an organized manner, using appropriate facts and relevant, descriptive details to support main ideas or themes; speak clearly at an understandable pace.
CCSS.ELA-Literacy.SL.5.4 - Report on a topic or text or present an opinion, sequencing ideas logically and using appropriate facts and relevant, descriptive details to support main ideas or themes; speak clearly at an understandable pace.

Materials-

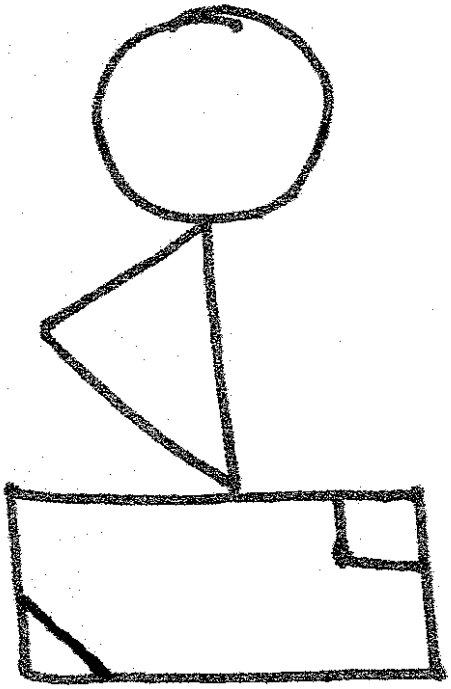
Teacher	Student
Worksheets with pictures on one side and space to write directions on the other side (cut in half to separate Picture A from Picture B). Projector to show class drawings (optional)	Paper with Picture A and spaces for written directions to ½ of class. Paper with Picture B and spaces for written directions to the other ½ of the class. Pen/pencil

Learning Tasks-

(Total Approx. Time: 15-30 minutes)

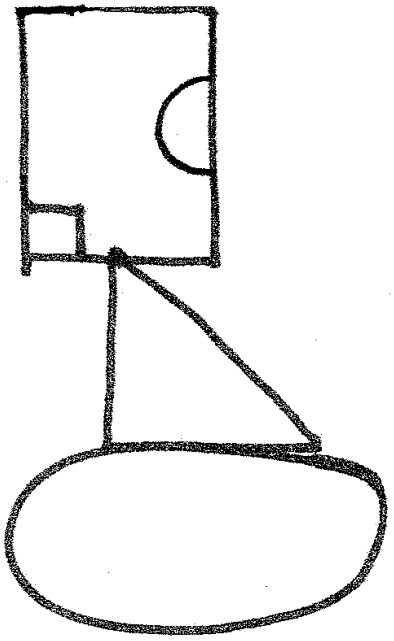
Approx. Time	**Learning Tasks**
0-5 min.	Divide the class into two groups. Give one group Picture A and Picture A Instructions and the second Picture B and Picture B Instructions. Then, task each student with writing simple "how to" instructions for someone in the other group to follow on the lines provided in the "Picture A/B Instructions".
5 min.	Have students cut or tear the picture away from their instructions. Collect all instructions from each group and randomly distribute these to the other group of students (students with picture A will get instructions for picture B and vice versa). Now, task students with creating a drawing based on the instructions provided.
5 min.	After students create the pictures, show both pictures to the entire class. Pair students up (one from each group) to discuss what could have been improved about the directions. They may reference the drawings were created from their directions.
5 min.	As a class, ask students to share about the activity. Prompt them to consider: What aspects of the instructions helped them accurately complete the drawing? What aspects of the instructions could have been improved? Students will like start to introduce how instructions could be more specific, such as how big to draw a circle, which way to orient a triangle, etc. Then, ask the same pairs of students to collaborate and edit one another's directions,
5 min.	Finally, present the anchor chart (see attached) and explain the elements of procedural writing. Now, ask students to write directions for Picture C using the elements of procedural writing.

Picture A

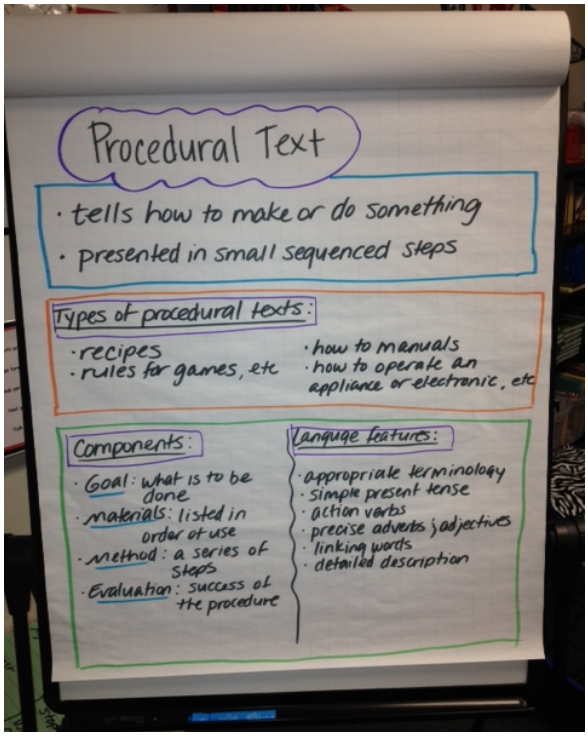


Picture A Instructions

Picture B



Picture B Instructions



Procedural Writing

The purpose of a procedure is to tell the reader how to do or make something.

The information is presented in a logical sequence of events, which is broken up into small sequenced steps.

Types of Procedural Texts
Texts that instruct how to do a particular activity: recipes, rules for games, science experiments, road safety rules, how to do it manuals.

Texts that instruct how to operate things: how to operate an appliance, a machine, a photocopier, or computer.

Components of Procedural Writing

A procedure usually has four components:

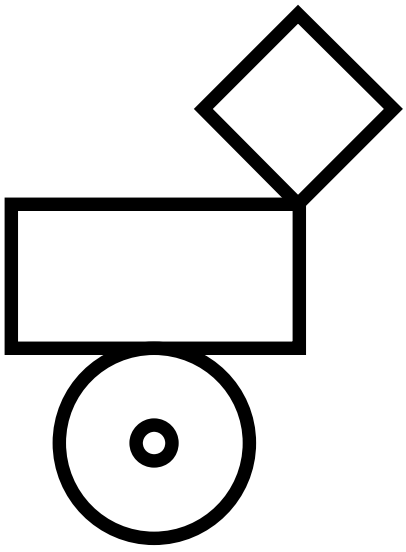
1. *Goal* – states what is to be done
2. *Materials* – listed in order of use, includes items needed to complete task
3. *Method* - a series of steps
4. *Evaluation* - how the success of the procedure can be tested

Headings, subheadings, numbered steps, diagrams, and photographs are often used to help clarify instructions.

Language Features of Procedural Writing

- Appropriate terminology
- Simple present tense (*Stir the mixture until it boils*).
- The reader is often referred to in a general way (*you cut, one cuts, cut*).
- Action verbs *(cut, fold, twist, hold etc)*
- Precise use of adverbs or adjectives *(slowly unwind)*
- Linking words to do with time *(first, when, then)* are used to connect the text
- Detailed information on how *(carefully, with the scissors)*; where *(from the top)*; when *(after it has set)*
- Detailed factual description *(shape, size, color, amount)*

Picture C



2: Sequencing

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces students to the KELP-CS programming interface, which we call LaPlaya. LaPlaya runs through an internet browser and uses block-based programming. In LaPlaya, students program sprites with scripts. **Sprites** can be images of animals, people, or inanimate objects. **Scripts** are a short program or collection of blocks that instruct a sprite how to behave and under what conditions to react. Scripts are constructed of blocks that are dragged onto the scripts area and snapped together.

This activity allows students to get familiar with creating scripts (dragging from the block list into the scripts are) and running scripts (resetting the program then pressing the green flag). They also learn that scripts must start with a block that says when to run (an Event block). The rest of the instructions must be placed in the proper order.

Objectives-

Students will be able to:

- Drag and drop blocks into programming areas.
- Combine multiple blocks to create scripts.
- Identify different areas of the LaPlaya interface (maneuver through block categories).
- Recognize that a script's action takes place on the stage.
- Learn that sprites are programmable agents.
- Organize blocks in the correct order to create a simple script.
- Run a script using the green flag button.
- Create a line on the stage using the pen down block.

Vocabulary-

Stage: Picture with the sprites on it where all of the action occurs
Sprite: A picture on the stage that you can control using scripts
Sprite List: Area of LaPlaya where you select what sprite you are programming.
Block: One command for a sprite to follow
Program: A set of instructions that are given to a computer to follow
Block List: Shows all available blocks in a lesson. Organized by block type such as motion, looks, and events.
Script: A very short sequence of blocks (instructions) for a sprite to follow
Scripts Area: Area of LaPlaya where active scripts are created.

New Blocks -

Block	Block Name	Block Description
	When Green Flag Clicked	Event block that starts a script. Script runs when a user clicks on the green flag

		button.
	Glide [ ] to [ ]	Motion block that moves a sprite with a speed (slowly, normally, quickly) to a specific sprite.
	Clear	Pen block that removes all pen lines.
	Pen down	Pen block that starts a drawing.
	Pen up	Pen block that ends a drawing.
	Set pen color to __	Optional: Pen block that changes the pen color.

Standards Emphasized-

Materials-

Teacher	Student
Computer Projector (or other way to demonstrate to students)	Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	**Learning Tasks**	**Purpose**	**Student Directions**
5-10 min.	*Task 1* Starting a Roof	Teacher uses task one as a demonstration to introduce students to LaPlaya. Teacher shows students different parts of the interface (sprites, scripts, stage, sprite list, blocks, block list, scripts area) and shows how to create a script. Students will then complete the task on their own at the computer.	In this task, you are learning how to make LaPlaya draw a line between two sprites. The script you need to make is already shown. You just need to make it yourself. 1) Click on the pen in the sprite list (if it is not already chosen) 2) Look for the proper blocks to put into the script by clicking on "Motion" and "Start" category buttons. 3) When you find the block, click and drag it onto your script area. 4) Once you have the scripts created, you are ready to run it. 5) Click on the "Get Ready" button (which sets up the program correctly) to get everything ready.

			6) Click on the "green flag" button to make it go.
	Task 2 Building a Roof	This builds on the first project. Now that students have drawn one line, they need to draw of a triangle. They draw a line to connect dots from 2 to 3 to 1.	Now you are ready to make a triangle - the roof of the house! You already completed one line. Now do the rest. Make sure you draw in the right order!.
	Task 3 Building a House	In this part, students finish the house and get a fun completion screen.	Now, using what you learned in the first two parts, finish the house!
	Task 4 - Bonus Draw a Cat	In this task, students are given a connect-the-dot script that is not quite in order. Once they run the program (click on green flag), they need to determine how the dots are out of order, and then fix the script.	Uh, oh! Someone did not get this script quite correct. Run this script by clicking on the green flag, and look closely how the drawing is made. Then go and fix the script so it will draw a cat!

Suggestions for Students who Finish Early-

- Bonus Task #4: Draw a Cat
- Play in the Sandbox (an open-ended version of LaPlaya that includes all of the blocks that students will be learning in this module).

Common Student Errors -

Task 2-
They should go to orange then pink, but rather they delete the orange dot script and replace it with a new script that goes to the pink dot. This results in an incomplete script that will only ever go to one dot or the other, but not both.

Students may also create a race condition if they use multiple "when green flag clicked" scripts to start the drawing to both pink and orange dots.

Teaching Hints -

You should demo Task 1 to your students first and go over what the different parts of LaPlaya are (the stage, the sprites, the blocks, etc.) and then go through what blocks they will be using (also that white scripts are inert) and what the goal of the task is before actually going onto the computer. You can also demonstrate dragging blocks over into the scripts area for class then have them do it on their own. Include instructions on what to do when finished (click the green check mark and it will tell you congratulation if you are finished or provide help if you are not).

3: Introducing Engineering Design Thinking

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces Engineering Design Thinking to students, which they will be using for the remainder of the module to design and create their own computer program telling a digital story. It is important for students to understand the importance of this design process and how it is used in many fields, not just engineering.

Objectives-

Students will be able to:

- Notice that the process of Engineering Design Thinking is much like the scientific process and is used in a wide range of fields (science, engineering, computer science, product design, etc.).
- Become familiar the steps in the engineering design process and will start to think about how they can apply them in designing their own computer program.

Standards Emphasized-

CSTA Computer Science Standards:
L1:6:CT- Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and texting) & Make a list of sub-problems to consider while addressing a larger problem.
L1:6:CL- Identify ways that teamwork and collaboration can support problem solving and innovation.

Materials-

Teacher	Student
Projector or something else to show the design process to whole class. (optional)	Student worksheet titled "Engineering Design Thinking" Design Notebook (optional)

Learning Tasks -

(Total Approx. Time: 15-20 minutes)

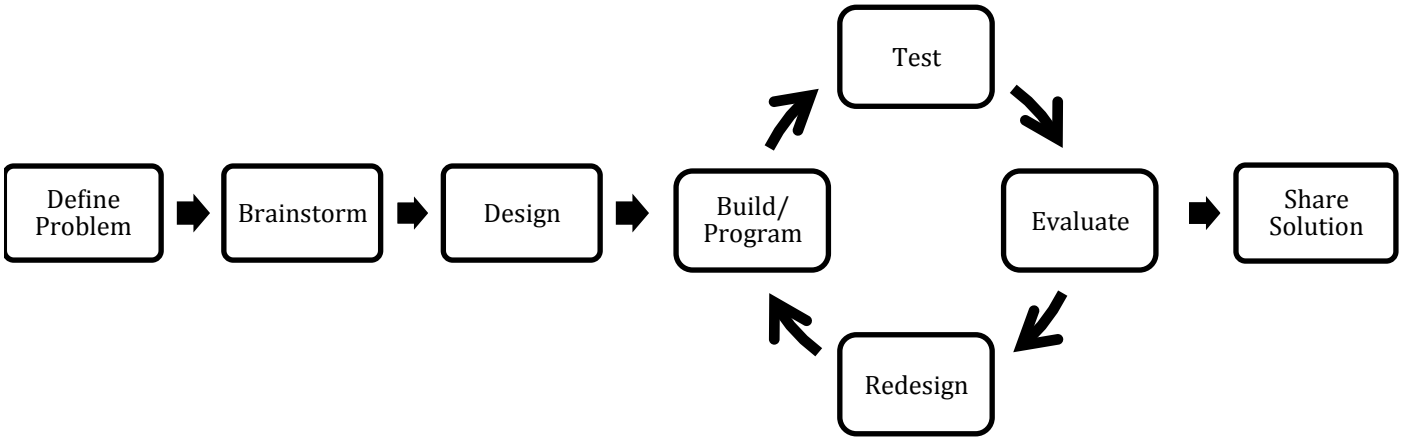
Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Introduce the process of Engineering Design Thinking and discuss with students how people in many different fields may use this process (scientists	The process of Engineering Design Thinking is important not only in science and engineering (as is indicated in the Next Generation Science Standards), but also in a wide array of fields that students should

	doing experiments, engineering solving problems, product designers creating a new piece of technology, clothing, toy, etc., architects designing a building, computer programmers creating a new program or app).	be aware of. Design Thinking is a powerful tool that can be used to solve simple or complex problems in many contexts and is a valuable skill for students to understand and master. Ask the students to think of ways the design process can be used for other things (they may think of novel ways they to use this process themselves like trying to reach something high or building a fort).
10-15 min.	Use a simple example of a problem that can easily be solved *(make an alarm for your bedroom door)* and go through the steps of the design process with this example. **Define the problem**- *Someone is stealing your allowance so you want to keep it safe in your room. The* ***constraints*** *are that you only have what is in your backpack to make it out of.* **Brainstorm**- *I need to make an alarm for my room! What can I do with my materials? Put a string across the door and attach something that will make noise to it.* **Design**- *Draw up a plan to figure out what materials you need, how much of them, and where they will all go.* **Build**- *Build the alarm.* **Test**- *Try out the alarm and see if it works on you first.* **Evaluate**- *What does/doesn't work with the alarm? What needs to be fixed?* **Redesign**- *Add things you think you may need, fix what didn't work, make your alarm better!* **Share**- *Ask your friend to come test out the alarm and see if they have suggestions for improving it.*	By going through the process with the class using a fairly simple engineering problem students will be able to work though each step and will get a fuller understanding of the Engineering Design Process. By doing this they will be able to transfer their knowledge and experience with the design process to the new context of computer programming using LaPlaya in the future. You may want to briefly discuss how computer programmers use this design process (like they will be doing). Go over how instead of building something physical (an alarm, machine, toy, building, etc.) programmers use the computer to build programs using code. They run those programs to test them and find any bugs (evaluating) before redesigning and fixing their programs to share with others.

Name/Number: _____

Engineering Design Thinking

What is **Engineering Design Thinking?**	The process ENGINEERS (and others) use to come up with SOLUTIONS to PROBLEMS.	
What is the process of **Engineering Design Thinking?**	**Define a Problem**	You or somebody else identifies a problem that needs to be solved. You also identify constraints that you must work within.
	Brainstorm Ideas	Working alone or with others you come up with many ideas to solve the problem and narrow them down to find the best one.
	Design	Planning what you will do so solve the problem. *(This includes storyboarding and creating a flow chart of your computer program.)*
	The Application Cycle	Building/Programming *(on the computer)*, Testing *(Trying it out)*, Evaluating *(what does/doesn't work)*, and Redesigning *(making it better)*. This is a cycle that you will go through multiple times before creating a finished solution to the problem.
	Sharing your Solution	It is important to share you solutions with others not only so they can benefit from them, but also so they can provide feedback to help create an even better solution.



Breaking Down Actions Fired Up

Teacher Lesson Plan

Lesson Rationale-

It is actually not easy to make a robot act like a human! Robots, like computers, interpret directions exactly in the form they are given. They do not guess what someone means to say. They do exactly what you tell them, and do not act without specific instructions. In addition, computer commands tend to be much simpler than the types of directions 4th graders typically use. The purpose of this exercise is to strengthen students' skills in breaking down actions into their smaller parts. The end goal is for students to recognize the need to break down actions and be able to do so when necessary in their programming projects.

Objectives-

Students will be able to:

- Develop the ability to break a complex task into a sequence of simple tasks
- Develop the ability to sequence information logically.

Standards Emphasized-

Materials-

Teacher	Student
North, South, East, and West labels for each wall of the classroom.	Robot Directions handout (either version) Pen/Pencil

Learning Tasks -

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks
0-5 min.	Divide the class into pairs of students who sit far apart from each other (Partner A and Partner B). The purpose is to produce a set of instructions that will lead one student from his/her desk to the door. Label each of the 4 walls in your classroom with the directions "North, South, East, and West" for students to reference.
10 min.	The pair goes to Partner A's desk. Partner B creates directions for Partner A using the Robot Directions Handout (they only need to fill in the number of steps to go and circle the direction). Partner A follows them **slowly** first with eyes open, then when they believe it is correct, with eyes closed.

10 min.	The pair then goes to partner B's desk. Partner A creates directions for Partner B. Partner B follows these **slowly** first with eyes open, then when they believe it is correct, with eyes closed.
5 min.	Discuss how students solved the problem. Ask how they moved partners in a new direction – did they ask their partner to turn or point in direction. Then discuss the process and proposed solution. Encourage students to test their ideas and modify their directions. This may take multiple rounds! Everyone is not expected to provide good directions on the first try. It is an **iterative** process, and mistakes lead to learning.

Fired Up: Robot Directions

		Circle one				*Write #*	
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**
Point Toward…	West	North South	East	&	**Walk…**	________	**Steps**

Fired Up: Robot Directions

		Circle one				*Write #*	
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**
Turn…	Left	Right	Around	&	**Walk…**	__________	**Steps**

4: Breaking Down Actions

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces students to breaking down actions. In our everyday lives, we act in specific ways, often in a specific order without realizing it. For example, when you brush your teeth you usually put the toothpaste on the toothbrush before cleaning your teeth (not the other way!). In programming, computer scientists break down larger action in to small, discrete pieces as well. They also pay particular attention to the order of these actions. When a program runs, the pieces of code they create occur in a specific order, from one command to the next. This activity prompts students to break down the movement of a sprite into a series of blocks, by specifying both the direction and the type of motion.

Objectives-

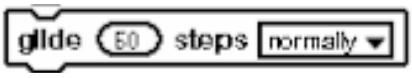
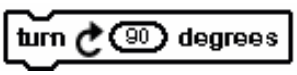
Students will be able to:

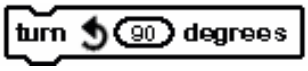
- Create a script that moves a sprite in a direction.
- Recognize that some movement can be done with fewer blocks (three glide blocks of 50 steps is the same as one glide block with 150 steps)
- Change the number of steps a sprite can make by typing what they want directly into the block.
- Recognize that there are multiple ways to solve a problem.
- Begin to understand that some solutions are more effective than others.

Vocabulary-

Sequence: A particular order in which events, movements or other things follow each other
Sprite: A picture on the stage that you can control using scripts
Script: A very short sequence of blocks (instructions) for a sprite to follow
Block: One command for a sprite to follow

New Blocks -

Block	Block Name	Block Description
	Glide ___ steps [normally]	Motion block that moves a sprite a variable length, such as 50 steps, with a specific speed (slowly, normally, quickly)
	Point in direction [ ]	Motion block that orients a block in four ways: right, left, up, and down.
	Turn __ degrees right	Motion blocks that rotates a sprite 90 degrees right.

	Turn __ degrees left	Motion blocks that rotates a sprite 90 degrees left.

Standards Emphasized-

Materials-

Teacher Computer Projector (or other way to demonstrate Task 1 to the class)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
	Task 1 Honey Walk	Teachers use this task to demonstrate how to program sprites to move. In LaPlaya, the movement of a sprite needs direction and a specific type of motion. Following the teacher demonstration, students complete tasks on their own at their computers.	Help the bear get to the honey! The script you need to create is shown. Look in the block categories and find the blocks you need to create the script. Each square on the screen is 50 steps.
	Task 2 Honey Hike	Students program a bear sprite to move across three grid blocks to the honey sprite.	Help the bear get to the honey! This time, the bear needs to get farther. How many steps is it?
	Task 3 Honey Climb	In this task, students need to make the bear go vertically. This requires a turn then glide. The solution is provided for them, and they need to copy it and run it.	Now the bear needs to climb up to the honey pot! The correct script is shown for you. Notice that you need to turn the bear before having her glide to it.
	Task 4 Honey Path	Now the students need to combine both skills - turning and navigating. The path is an L shape.	Help the bear get to the honey! Figure out how to navigate the bear to the honey
	Task 5 Honey	In this task, students are asked to find what hidden things happen. This is to point out that there are	Just as in the last project, figure out how to get the bear to the honey.

	Adventure	more scripts going on than they see and to emphasize that this was done as a design decision for them, the user. The only scripts being hidden are complex scripts, ones that we don't want them to worry about. But we still want fun things to happen. So, their job is to find them! They should complete a very short list like this: Sprite X does Y when Z happens Honey pot tips over when bear hits it Honey pot says "Good job" when Bear hits it Bushes say "" when Bear hits them	Don't hit the bushes!

Suggestions for Students who Finish Early-

- Invite student to practice the activity by using both relative and absolute direction change.
- Play in the Sandbox!

Common Student Errors -

Students may try to reprogram things that are already programmed if they are unlocked

Teaching Hints -

5: Defining an Engineering Problem

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces students to the engineering problem that they will be solving through the programming of a digital story. The teacher gives students a topic that is phrased as en Engineering Problem (California Missions, a book report, a science demonstration, etc.) that students will need to create their program within and other constraints (the LaPlaya programming environment, time, etc.). Students will then write a problem statement about what the problem is, and why it is important to solve.

Objectives-

Students will be able to:

- Write a problem statement that includes who has a problem, what the problem is, and why it is important to solve.
- Identify constraints for their project.

Standards Emphasized-

CSTA Computer Science Standards:
L1:6:CT- Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and texting) & Make a list of sub-problems to consider while addressing a larger problem.

Materials-

Teacher	Student
"Engineering Design Thinking" worksheet (for reference) Computer/Projector to show example projects	"Engineering Problem and constraints" worksheet Pencil or pen Design Notebook (optional)

Learning Tasks -

(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
2-3 mins.	Use the "Engineering Design Thinking" worksheet that students were given previously as a reference to introduce this first step in the Design Thinking process; Define a Problem.	By referring back to what students have already gone over it reinforces the parts of the design cycle and contextualizes them with the visual provided on the "Engineering Design Thinking" worksheet. Going over the new vocabulary is also essential for helping

	Connect this with the definitions at the top of their Engineering Problems and Constraints" worksheet.	students to understand the parts of an engineering problem and differentiating the constraints they must work within.
3-5 mins	Introduce the Engineering Problem that you have decided for you class to work on. Some suggestions include: • A younger class wants to know about natural disasters. How can we share what we know with them? • I (the teacher) want to know what happens in the book James and the Giant Peach, but I don't have enough time to read it. Can you summarize it for me?	By introducing an engineering problem in this way it gives students more of a purpose for creating their digital stories. They also include all of the elements necessary for students to complete a problem statement: • Who has a problem • What is the problem • And, Why is the problem important You may have your students create a digital story for any sort of engineering problem (it is up to you), but the 3 parts of the problem statement (see above) should be clear to students.
3-5 mins.	Have students complete #1 – 3 of the problem statement on their "Engineering Problems and Constraints" worksheet.	This is the first, and most important part of defining an engineering problem, and students can and should refer back to this throughout the creation of their programs.
5-10 mins.	Help students identify the constraints they will need to work within when solving this engineering problem (They must use LaPlaya, they will have only a set amount of time, etc.) Students can then complete #4 of the problem statement on their worksheet.	The basic constraints include the LaPlaya programming environment and time (which is up to you), but students may also come up with interesting ideas for constraints that may be appropriate. Helping students with this process is essential because these limitations should be explicit before they begin working on the more open-ended programming environment so they don't get lost in the many options available.
3-5 min.	Go over with students how they will be recording their data and progress throughout the design thinking cycle of programming their digital story (this is up to you).	It is crucial that students get into the habit of recording things and keeping track of what they have done, need to do, and will do. You may decide to have you students create a design notebook or simply a binder of the worksheet provided with some extra paper; How they record things is up to you.

Name/Number: _____

Engineering Problems and Constraints

What is an engineering **problem**?	Something that you, another person, or a group of people NEED. Sometimes you might identify a problem yourself or be asked by someone else to help solve a problem they have identified.
What are **constraints**?	LIMITATIONS that you must work within while trying to solve an engineering problem. This could be time, money, materials, or knowledge.
How do you write a **problem** statement?	1. WHO has the **problem** or need? 2. WHAT is the **problem** or need? 3. WHY is it important to solve (the **goal**)? 4. What **constraints** will I need to work within?

For the digital story you will be creating with LaPlaya your teacher will ask you to help solve a problem that they have identified.

Fill out the problem statement below.

1. WHO has the **problem** or need? *(Your teacher, you, someone else?)*	_____ _____
2. WHAT is the **problem** or need? *(Your teacher will tell you this)*	_____ _____
3. WHY is it important to solve? *(What is the **goal** of the project?)*	_____ _____
4. What **constraints** will I need to work within? *(Time, Knowledge of Programming, LaPlaya limitations, something else?)*	_____ _____ _____ _____

6: Event-Driven Programming: When Sprite Clicked

Teacher Lesson Plan

Lesson Rationale-

In this lesson, students create a program that responds to a user, specifically when a user clicks on a sprite on the stage. To create a fun game, computer scientists can create programs that require a user do something such as click something with his or her mouse or press a button. These actions are called events. In LaPlaya, students start a script with event blocks (brown blocks, with a curved top). Though they may not have realized it, they already used event blocks in the last two activities (when green flag pressed). In this activity, students will learn how to create a script that runs when a user clicks on a sprite on the stage.

Objectives-

Students will be able to:

- Change the relative size of sprites.
- Engage a user in their programs.
- Create scripts that run when a user interacts with the stage.

Vocabulary-

User: The person playing the game, running the program, etc.
Control Blocks: Blocks that determine when something should happen.
Event: Something that the user does (click sprite, press button, etc.)
Interactive: A program that responds to things the user does.

New Blocks -

Block	Block Name	Block Description
	When Sprite is clicked	Event block that runs a script when a user clicks on a specific sprite on the stage.
	Increase size by ___	Look block that will make a sprite bigger by a variable increment (student types in amount of change).
	Decrease size by ____	Look block that will make a sprite smaller by a variable increment (student types in amount of change).
	Glide __ steps [right]	Motion block that moves a sprite a variable increment in a direction (left, right, up, or down)
	Say ___ for __ sec	Look block that creates a pop up text

		bubble above a sprite that stays on screen for variable amount of time.
	Think ___ for __ sec	Look block that creates a pop up think bubble above a sprite that stays on screen for variable time.

Standards Emphasized-

Materials-

<u>Teacher</u> Computer Project (to demonstrate task 1)	<u>Student</u> Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	**Learning Tasks**	**Purpose**	**Student Directions**
5 min.	*Task 1* Growing Apples	Teachers use this task to demonstrate how to create a script using the "When Sprite is Clicked" event block using a program that changes the size of apple sprites. Following the teacher demonstration, students complete the task independently on their computer. Remember that there are two parts to every script: 1) <u>When</u> you want it to happen (Events) 2) What you want it to <u>do</u> (Motion, Looks, or other blocks)	Program each green apple sprite to get bigger when they are clicked. Hint: Use "increase size by" Remember that there are two parts to every script: 1) <u>When</u> you want it to happen (Events) 2) What you want it to <u>do</u> (Motion or Looks) The script you need to create is shown for the first apple.
	Task 2 Rotten Apples	In this task, students are asked to make the rotten apples fall to the ground when they are clicked. This is to reinforce the "on sprite clicked" that they learned in the previous task along with the motion blocks they learned in the prior lessons.	Oh, no! Some of the apples are rotten! Program a script in the rotten apples that will make them fall to the ground when clicked. Remember that scripts have two parts: 1) <u>When</u> they happen (events) 2) What to <u>do</u> (motion / looks
	Task 3	This task reinforces the "on sprite	Your friend cannot remember

	Planets	clicked" block and introduces a new block - the "Say" block. Students program each planet say its name. The name is provided in the sprite list so that students have a guide for spelling. Students need to program at least 4 planets to complete this project. If students want to, they can make the planets do fun things when they finish. For example, the planet could zoom around the screen, get bigger and then smaller, turn around in a circle, etc.	all the planet names and wants something to help him or her learn them again. In this project, you will use a new block, the "Say" block. For every planet, when the user clicks that planet, program the sprite to say its name. Don't worry if you have forgotten the names. The labels of the sprites say the names. Look at the sun to see how it's done!
	Task 4 Bonus Piano Part 1	In this optional project, students program a piano where the keys are sprites. Students write scripts so that when each key sprite is pressed, it plays its note.	You can make a piano with LaPlaya! There is a new block under "sound" that plays different notes. All of the keys are sprites. Program each key sprite to play the correct sound when it is pressed! The C key has a picture of the script for you. Hint: Match the sprite name to the key sound.

Suggestions for Students who Finish Early-

- Bonus Task 4
- Play in the Sandbox!

Common Student Errors -

- Growing apples task 1: Students may use a set size (##%) block (either alone or in addition to an increase size block) so the apple grows all at once and doesn't "need" to be clicked multiple times to "grow" when clicked.
- Planets task 3: Students may not realize that they need to create separate scripts for each sprite (each planet) to have them all say their name. To do this they need to click on each sprite in the sprite list and program in their respective scripts area.
- Students may try to reprogram things that are already programmed if they are unlocked or delete scripts that they already programmed thinking that they must delete it before moving on.

Teaching Hints -

- Students may forget to start a script with an event block. To help them remember, tell them that a finished script looks like the top-half of a hamburger. The brown event blocks is the top bun, and the other blocks are the layers of the burger.
- Students can copy/paste scripts from one sprite to another (useful in the planets task!). To copy a script, drag the script over to the other sprite in the sprite list (bottom right corner).

7: Brainstorming Digital Stories

Teacher Lesson Plan

Lesson Rationale-

In this activity students will be going through the brainstorming process to generate ideas for a digital story and then narrow them down to find the best solution. This process teaches students that all ideas are good ones, and those that may sound crazy might be the best ones. This also makes students evaluate their own ideas and come up with a rationale to pick the best one. Students will also write a persuasive piece about their final solution and how it solved the engineering problem and fits within the constraints.

Objectives-

Students will be able to:

- Generate multiple ideas to solve the engineering problem presented to them.
- Use reasoning to narrow down their ideas and ultimately pick the best one they came up with.
- Create a piece of persuasive writing about their best idea to solve the engineering problem and address constraints.

Standards Emphasized-

Common Core State Standards:
ELA-LITERACY.W.4.1- Write opinion pieces on topics or texts, supporting a point of view with reasons and information. Introduce a topic or text clearly, state an opinion, and create an organizational structure in which related ideas are grouped to support the writer's purpose. Provide reasons that are supported by facts and details. Link opinion and reasons using words and phrases (e.g., *for instance*, *in order to*, *in addition*). Provide a concluding statement or section related to the opinion presented.
ELA-LITERACY.W.4.6- With some guidance and support from adults, use technology, including the Internet, to produce and publish writing as well as to interact and collaborate with others; demonstrate sufficient command of keyboarding skills to type a minimum of one page in a single sitting.

Materials-

Teacher	Student
"Engineering Design Thinking" worksheet (optional) Timer (optional)	"Brainstorming" worksheet "Persuasive Writing" worksheet Pieces of paper & pen/pencil Design Notebook (optional) Computer (optional)

Learning Tasks -

(Total Approx. Time: 45-60 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Introduce the Brainstorming part of the Engineering Design Process (optional to refer back to the "Design Thinking" worksheet visual) and go over the definitions on the "Brainstorming" worksheet and the parts of the brainstorming process (which is iterative).	Introducing the vocabulary and the contexts that brainstorming is used in will help solidify what the process is to students and going over the parts of the process before students actually engage in it themselves should give them an idea of what is expected before beginning.
5-10 min.	Discuss guidelines for step 1 of the brainstorming process; Generate as many ideas as you can as quickly as you can. Emphasize that there are no bad ideas and that students can write or draw ideas on their paper, but they don't need to be pretty or descriptive at this point. You may want to time students and challenge them to come up with the most ideas at the end of that time. We suggest about 5 minutes for this 1st round of idea generation.	In this step it is important that students understand that their ideas, writing, and drawings of those ideas do not need to be too descriptive or perfect. It is important that students come up with many ideas for their digital stories so they have a larger pool to chose from when picking their final, best idea to create their program on. Students can be messy when writing and drawing their ideas on their paper, what it important is that they know what the ideas are so they can evaluate them in the next step of the process.
3-5 min.	Have students evaluate their ideas and write 3 to 5 of their best ones on the "Brainstorming" worksheet.	By doing this students will need to take into consideration the engineering problem and constraints when picking their best ideas. They may refer back to the "Engineering Problem & Constraints worksheet.
3-5 min.	As a class pick an idea that was generated (use this as an example) and make that idea better and more elaborate to demonstrate step 3 of the brainstorming process; building on ideas.	Building and elaborating on their ideas may be difficult for students so going through this process as a whole class may help them understand how to do this and will help them when they do step 3 of the brainstorming process on their own.

5-10 min.	Have students build and elaborate on their top 3-5 ideas (from step 2) on a new sheet of paper. You may want to walk around and help students with this process if it appears that they are having difficulty.	This process helps students build on ideas and think about how they can make ideas better. This skill is important and can be applied to other people's ideas as well. You may want to discuss that this is what is done all the time; people build on other's ideas and make them better (ex. The iPhone used others' ideas about touch screens and combining internet/phone capabilities and put them together in a new, better way).
3-5 min.	Have students go through and evaluate their ideas and pick the single best one. Students will then write this down on step 4 of their "Brainstorming" worksheet.	Once again, students are having to take into consideration the engineering problem and constraints when evaluating their own ideas and narrowing them down to the best single idea they generated. This will be the idea that they will create their digital story on using LaPlaya throughout Module 1
5-10 min.	Have students get out their "Persuasive Writing" worksheet and go over what persuasive writing is, what the parts of a persuasive piece are (introduction, body text, conclusion) and explain that they will need to persuade someone else that their idea for their digital story is a good one that solves the engineering problem and fits within the constraints.	Having students learn about and write a persuasive piece is a big part of the common core standards for writing and is important in the engineering design process because it is critical that you are able to persuade others that your ideas are valuable. This skill is used in many fields and in everyday life (ex. If you want to go to Disneyland you need to persuade your parents to go).
5-10 min.	Students will complete the persuasive writing prompts on their "Persuasive Writing" worksheet before creating a finished persuasive piece online. or using paper and pencil.	Persuasive writing is an essential skill for students to practice and learn. If students complete their writing on the computer they will also be practicing valuable typing skills and be getting more comfortable using technology. You may extend this lesson by having students "pitch" their ideas to the class after they complete their persuasive writing.

Name/Number: _____

Brainstorming

What is **brainstorming**?	A technique to generate IDEAS and come up with CREATIVE solutions to problems. It started in advertising and is now used for business, research, writing, design, and much more.	
What are the parts of **brainstorming**?	**Step 1:**	DRAW or WRITE as many ideas as you can think of as quickly as you can. *They don't have to be pretty or detailed, as long as you know what it means.*
	Step 2:	Pick 3 – 5 of the BEST ideas you came up with. *Write those at the top of a new page.*
	Step 3:	Use the ideas you picked in **Step 2** to come up with more ideas. Make them BETTER and more detailed.
	Step 4:	Pick the BEST idea you came up with to create your Design Thinking project.

The process of brainstorming:

Step 1: Think of as many ideas as you can for your digital story. Keep the project guidelines in mind when you generate ideas.
Write on a blank sheet of paper.

Step 2: WRITE the best ideas you came up with on the lines below:

_____

_____

_____

_____

_____

Step 3: *How can you make those ideas better?*
Can you combine any of the ideas?
Can you add to an idea to make it more interesting?

Step 4: What is the BEST idea you generated?
This will be your Design Thinking project.

_____

Name/Number: _____

Persuasive Writing

What is **persuasive** writing?	Writing about your OPINION on something and supporting your point of view with EVIDENCE (information, facts, reasons).	
What are the parts of a **persuasive** piece?	**Introduction:**	Introduce the topic and state an opinion.
	Body Text:	Group related ideas together and support your opinion with facts and details. Use phrases like "for instance", "in order to", or "in addition".
	Conclusion:	Summarize your argument and state why the evidence you provided supported your opinion.

Engineers, Scientists, and many others have to PERSUADE people that their idea or opinion is good and should be used to create a product, solve a problem, or make decisions. To do this they must make a good argument with EVIDENCE that will convince people to share their view.

Writing a persuasive piece about your digital story:

What was the idea you picked to use for your digital story?

_____

_____

Why is this a good idea for this project?
(1 sentence only)

_____

_____

What is your evidence that this is a good project idea?
(How does it fit what the problem and constraints were? Will you be able to program it using LaPlaya? What else makes it good?)

_____

_____

_____

_____

_____

Summarize your whole argument in 1 sentence.

_____

Now you have some ideas to start WRITING your **persuasive** piece!

8: Storyboarding Digital Stories

Teacher Lesson Plan

Lesson Rationale-

In this lesson students will be elaborating on the idea for their digital story and creating a storyboard that will not only help them determine the structure of their story (including an introduction, memorable moments, and a conclusion), but will also help them communicate their planned story with others. Students will also learn the basics of sketching and communicating ideas efficiently by finding the balance between providing enough detail without wasting too much time.

Objectives-

Students will be able to:

- Create a basic storyboard of their proposed digital story that includes basic sketches of main events and short written descriptions.
- Discover that sketching does not include excessive detail and should not take too much time.

Standards Emphasized-

Common Core State Standards:
CCSS: ELA-LITERACY.W 4.3: Write narratives to develop real or imagined experiences or events using effective technique, descriptive details, and clear event sequences.

Materials-

Teacher	Student
Computer Projector to show example(s) of storyboards (optional)	"Storyboards" worksheet "Sketching Basics" worksheet Storyboard Template(s) Pen/pencil Design Notebook (optional)

Learning Tasks -

(Total Approx. Time: 45-60 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Briefly go over what storyboards are (using the "storyboards" worksheet) and how they are used in comic books, movies, etc. (optional to show a storyboard to students that they may be familiar with (find some examples online).	It is important to go over the new terminology with students and providing an example gives them a familiar context to connect this new thing they are learning about to. If you show an example of a storyboard you may want to find one for a children's movie that students will most likely have seen before (Toy story, Frozen, etc.)

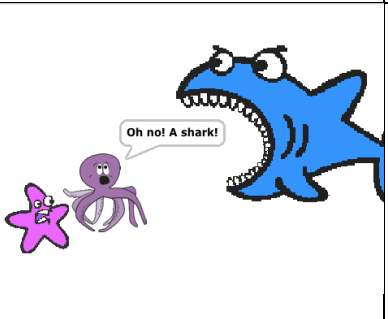
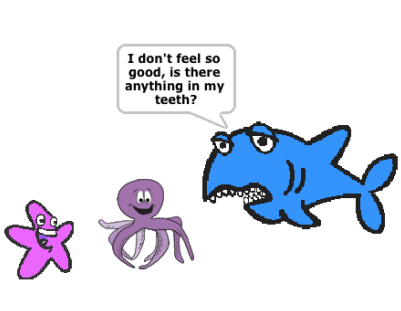
3-5 min.	Discuss with students the main parts of a story (introduction, memorable moments, conclusion) on the "Storyboards" worksheet and how to relate that to their digital story's storyboard.	Emphasize how the different parts of a story are essential to creating a good story that an audience will follow and enjoy. This process is also important in programming because you need to think about introducing a new user to your program, making it memorable throughout, and understanding how to wrap things up when the user is finished with the program.
3-5 min.	Go over the different parts of a storyboard with students and use the example storyboard provided for reference	The example storyboard, which uses screenshots from LaPlaya, gives students a good context for how their storyboard should look when finished.
5-10 min.	Go over the basics of sketching using the "Sketching Basics" worksheet and emphasize to students that detailed drawings are not necessary, it is more important to get your ideas across efficiently than spend too much time making detailed drawings.	The worksheet provides examples of different types of things that students may wish to sketch in their storyboards (people, animals, objects, and actions). Making sure that students know that their sketches don't need to be perfect or even look realistic (stick figures are great!) will help them understand that sketches of their ideas should be less detailed (as opposed to technical drawings in engineering, which need more detail). This should also make sure that students don't waste too much class time on this part of the design process.
3-5 min.	Have students do a basic sketch on their worksheet that includes a person or animal doing something and another object.	By having students practicing basic sketching with a generic prompt like this students will have an idea of how detailed they need to be in their storyboard sketches and will have practice showing actions in their sketches.
15-30 min.	Have students complete a storyboard for their own digital story. Their story will be about the best idea they came up with in the brainstorming process and wrote the persuasive piece on. After 5-10 minutes students should be moving on from one frame of their storyboard to the next (ex. from the introduction to the first memorable moment).	Creating a storyboard for their digital story is a huge part of the "design" portion of the Engineering Design Process and will be critical in later programming lessons when students learn about scene changes. You should monitor student progress throughout this storyboarding process to ensure that the students don't spend too much time on one part of their storyboard and end up not having time to finish. Students will be using these storyboards to create flowcharts in the next engineering design thinking lesson.

Name/Number: _____

Storyboards

What are **storyboards?**	*A series of sketches with written descriptions that represent the important parts of a planned program. They are often used for movies, television, and even comics.*	
What are the <u>parts</u> of a **storyboard?**	**Introduction:**	Set the scene for your story. Introduce your characters and establish the setting.
	Memorable Moments:	Something important that happens to your character (they go somewhere, do something, or meet someone). *You can have more than one memorable moment.*
	Conclusion:	Resolve any problems, conflicts, or events that took place in your story. May include a message for users (ex. "The moral of the story is…").

A storyboard has 2 main parts, the SKETCH of the main events in the story and the written DESCRIPTION of what is happening.
See the example below before creating a storyboard for your own story.

Introduction	Memorable Moment #1	Conclusion
		
Alga the Octopus and Marty the Starfish are friends with a nice life in the ocean.	A big shark shows up and scares the two friends with his mouth wide open.	The shark has a toothache and just wanted their help; he didn't want to eat them.

Name/Number: _____

Sketching Basics

What are **sketches?**	Quick drawings that are designed to show others what you mean without wasting time on too much detail. *(At this phase in the design process too much detail may be distracting).* They can include some writing as well.
What are the <u>parts</u> of a **sketch?**	**People:** 
	Animals: 
	Actions: 
	Objects: 

Do a simple SKETCH of either a person or an animal doing an ACTION. Include at least one OBJECT also→	

Figure adapted from: Greenberg, S., Carpendale, S., Marquardt, N., Buxton, B. Sketching User Experiences: The Workbook. Morgan Kaufmann, 2012.

Name/Number: _____ Page # ____ of ____

Storyboard for the story about _____.

Write the part of the story (Introduction, Memorable Moment #__, Conclusion) that each scene is describing on the line above it-

_____	_____	_____
Description: _____	Description: _____	Description: _____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

9: Event-Driven Programming: When Key Pressed

Teacher Lesson Plan

Lesson Rationale-

In this lesson, students continue creating programs that run on events triggered by a user. In the last lesson, students created scripts that ran when a user clicked on a sprite (using the event block "On Sprite Clicked"). Here, students create scripts with a new event block, "When Key Pressed." Keys are buttons located on the keyboard such as numbers, letters, or arrow keys. Students will complete multiple tasks by where they need to program a script to run on different computer keys.

Objectives-

Students will be able to:

- Change the absolute direction of a sprite (point up, down, left, or right).
- Write scripts to run when the four arrow keys are pressed.
- Correlate sounds to play on different keys.
- Engage a user in their programs.
- Create scripts that run when a user interacts with the keyboard.

Vocabulary-

User: The person playing the game, running the program, etc.
Control Blocks: Blocks that determine when something should happen.
Event: Something that the user does (click sprite, press button, etc.)
Key: Buttons located on a computer keyboard such numbers, letters, or arrow keys.
Interactive: A program that responds to things the user does.

New Blocks -

Block	Block Name	Block Description
	When [Key] Pressed	Event block that runs a script when a user presses a key on the keyboard.
	When [Get Ready Button] Pressed	Event block that runs a script when the blue "Get Ready" button is pressed. Typically the start of an initialization script.
	Point towards [ ]	Motion block that orients a sprite in the direction of another sprite.
	Play note [C] for __ beats	Sound block that creates a sound for a variable amount of time.

Standards Emphasized-

Materials-

Teacher Computer Projector (to demonstrate Task 1)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	**Learning Tasks**	**Purpose**	**Student Directions**
5 min.	*Task 1* Rocket	Use this task to demonstrate how to create a script with the "When [up arrow] key is pressed" that moves a rocket ship upwards. Following the teacher demonstration, students complete the task independently on their computer.	Help the astronaut blast off into space to explore the universe! Program a script that, when you press the up arrow, makes the rocket go up. A picture of the script is given to you for the rocket. Make your own copy of that script. Don't forget to click the "get ready" button and then the green flag to start!
	Task 2 California Geography	Students program a car sprite to move in four directions (up, down, left, and right) on the four arrow keys. Then they move the car with the arrow keys to travel to multiple California cities.	Program the car sprite to move in four directions (up, down, left and right) on the four arrow keys. Then, use the arrow keys to move the car sprite to the different California Cities! Hint: You should have four scripts, one for each arrow key!
	Task 3 Catching Fruit	Students program a basket sprite to move left when the left arrow key is pressed and right when the right arrow key is presses to catch apples as they fall. The apples fall when the student clicks on them.	Program the basket sprite to move left and right when the left and right arrow keys are pressed. Click on each apple and catch them in the basket as they fall!

	Task 4 Bonus Piano, Part 2	Students turn their keyboard into a piano by programming different keys to make different sounds when pressed. The selected keys match proper hand position for typing.	Turn your keyboard into a piano! For each note, write a script that will make it play the correct sound. Press the Green Flag to start! Hint: Use the script on Key 1 as a guide.

Suggestions for Students who Finish Early-

- Bonus task 4 (Piano Part 2)
- Play in the Sandbox!

Common Student Errors -

- Task 1 (Rocket): Students may not keep pressing the arrow buttons to make the rocket move (if they don't see something happening like they expected they may give up)
- Task 2 (CA Geography): Students may program the keys to go directly to a city (ex. When up arrow key pressed it goes directly to Santa Barbara) rather than programming the arrows to make the car move in that direction in small increments to "drive" to the cities (ex. When up arrow key pressed point in direction up and move 50 steps)
- Students may try to reprogram things that are already programmed if they are unlocked

Teaching Hints -

10: Thinking About the User

Teacher Lesson Plan

Lesson Rationale-

The **user** is what the person who plays with software is called. Students have been "users" of software throughout their lives; they are currently users of LaPlaya in this module! Computer scientists often want to engage users in their programs through events such as clicking the mouse or pressing the keyboard (like the last activity!). Sometimes, in order to make software easy to use, computer scientists also must run special programs in the background. This is true with LaPlaya! In all the activities for this curriculum, our computer scientists run invisible scripts to help students finish the tasks. In this activity, students begin learning the difference between being the **user** of software and being the **designer** of software. In these tasks, students will analyze finished LaPlaya tasks to determine what actions may be hidden and discuss how they can engage the user in their own programs.

Objectives-

Students will be able to:

- Differentiate between a user and a designer.
- Identify activities when they acted as a user and when they acted as a designer.
- Consider what types of scripts are hidden in LaPlaya.
- Discuss the benefits and consequences of hiding scripts in LaPlaya.

Vocabulary-

User: The person playing the game, running the program, etc.
Designer: Computer scientist or programmer than develops software.

New Blocks -

Block	Block Name	Block Description
	Place at [ ]	Motion block that immediately places a sprite at a particular location.
	Wait [ ] secs	Control block that adds a variable amount of time between actions.
	Repeat __	Control block that repeats scripts in a bracket. Repetitions by vary by any amount.

Standards Emphasized-

Materials-

Teacher Computer Projector (optional)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	**Learning Tasks**	**Purpose**	**Student Directions**
10 min.	*Task 1* Finding Hidden Scripts	In this task, students look through a completed project. They are supposed to play this project and identify what hidden things are happening for which they do not see scripts. The solution should be a list in this format: Sprite X does What when This Happens Bush says "" when bear hits it (if Bear hasn't hit bush): Honey pot says "Good Job" when bear hits it Honey pot tips over and bear goes inside when bear hits it	Your job in this task is to find all the hidden scripts! Some of the sprites do things, but you are not able to see those scripts. This is because we, the designers, do not want to confuse you, the user, with complex scripts. You need to produce a very short list that looks like this: Sprite _____ does _____ when _____ happens
	Task 2 Analyzing the User Interface	In this task, students are asked to figure out all of the things that cause actions in the program. They should check every key on the keyboard as well as click everywhere on the stage. Their solution list should look something like this: Key X causes sprite Y to do Something If I click sprite Z, then sprite L does Something	Here is a tortoise and the hare project. The tortoise and the hare are supposed to race but the designer did not make it very clear. Your job is to figure how a user can make actions happen in this program. Make sure you check all of the keys on the keyboard and click things on the stage. Your solution should look something like this: When I press key _____, sprite _____ does _____. When I click on sprite _____, sprite _____ does _____.

	Task 3 Designing the User Interface	This task starts with the same file as the previous task ("Analyzing the User Interface"). This time, students will fix the interface. We want them to choose an interface that makes sense. For example, clicking the two animals should do something similar (like make them both go). Or they could both go on the green flag (even better!). Likewise, keys h and b should do similar things (perhaps those would change the colors). Whatever they choose, they should be consistent in having the same type of command do the same thing.	Your job now is to fix the project you just explored. It was not a good interface to have one animal start when it was clicked but the other animal start when a key was pressed. Here is what you need to do: Make the animals start their race on the green flag. Make them change and get bigger then smaller when you click on them. Don't do anything when you press keys.

Suggestions for Students who Finish Early-

- Play in the Sandbox!

Common Student Errors -

Teaching Hints -

- You should facilitate all of these tasks and demonstrate them initially for students before having them work on their own. These tasks are designed to be a little hard because students are debugging the program and trying to figure out what it going on "under the hood".

11: Flow Charts

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces flowcharts to students, which are used by computer programmers (among others) to help plan out their program and visually display all of the different parts of a program. This lesson bridges the conceptual divide between the storyboards students created previously and the sophisticated flowcharts that students will be creating once they start programming and revising their digital stories.

Objectives-

Students will be able to:

- Identity the different parts of a flowchart and understand what each means.
- Interpret a basic flowchart.
- Create their own flowchart using their storyboards as a guide.

Standards Emphasized-

Materials-

Teacher	Student
	"Flowcharts" worksheet Flowchart template Storyboard template Design Notebook (optional)

Learning Tasks -

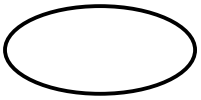
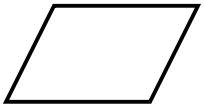
(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Go over what flowcharts are and how they are used in programming. Emphasize that all software developers (programmers) go through this process to layout what the parts of their program are and how everything works together before even getting on the computer.	Going over what flowcharts are and how they are used in computer science is important for students to understand the importance of this process in the engineering design cycle before getting on a computer to program their digital stories.

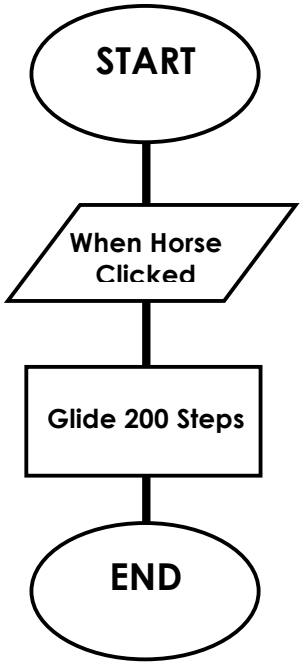
5-10 min.	Go through what the different parts of a flowchart are (lines, dotted lines, rectangles, ovals, and parallelograms) and go through the example flowchart given on the worksheet.	It is important to spend some time going though the different parts of a flowchart with students because they can be confusing and might not make sense out of context (not on the computer). As students complete the questions related to the example flowchart on their worksheets you may want to help them answer the questions given.
3-5 min.	Guide students through creating the basic flowchart on the left of their "Flowchart Template" worksheet. Students should use their storyboards (the description of each scene) as a guideline to help them fill out the three boxes provided in this flowchart.	This step of creating a basic flowchart using their storyboard as a guide is a way to let students transition more smoothly from thinking of their digital story as a visual or written story to thinking of it more like a computer program.
5-10 min.	Allow students to create a fuller version of their flowchart in the blank box on the right of the "Flowchart Template" worksheet. Students should look at the different shapes that denote different parts of a program and think about how they can incorporate those into this new flowchart of their program.	Students should start thinking of their digital story more like a program and begin figuring out what kinds of commands will need to be used to start the main events that will occur in their program. You may want to check in with students throughout this process and relate the parts of the flowchart to the things they want to happen in their digital story.
2-3 min.	Discuss how students can add different parts (new shapes and lines and even new flowcharts for each sprite) to their own flowcharts for their digital story when they learn more programming skills and elaborate on their program.	It is good to let students know that the flowchart they will be creating now is not their final one and that they will be learning more programming skills and adding to their digital story and flowchart throughout the design process. (ex. An INPUT is what makes the event happen→ an apple falls on a sprites head, which makes the sprite say "ouch)

Name/Number: _____

Flowcharts

What are **flowcharts?**	A way to organize and show the "flow" of your program. Programmers use flowcharts to work out the steps in a program they are designing before coding anything on the computer. You will create one for each sprite later on.
What are the parts of a **flowchart**?	 The START or END of a program.
	Shows the RELATIONSHIP between the parts of a program.
	 A PROCESS that will be performed. (Ex. "Glide 50 Steps")
	 An INPUT that starts a process. (Ex. "When Sprite Clicked")
	The RELATIONSHIP between different sprites (Ex. "When Sun Clicked" - - "Boy says It's hot!")

Interpreting a flowchart:



1. What does this program do?

_____

_____

2. What is the INPUT that starts the process "Glide 200 steps"?

_____

_____

3. What kind of a program do you think this flowchart is for?

_____

_____

4. How could you use this to help write a flowchart for your digital story?

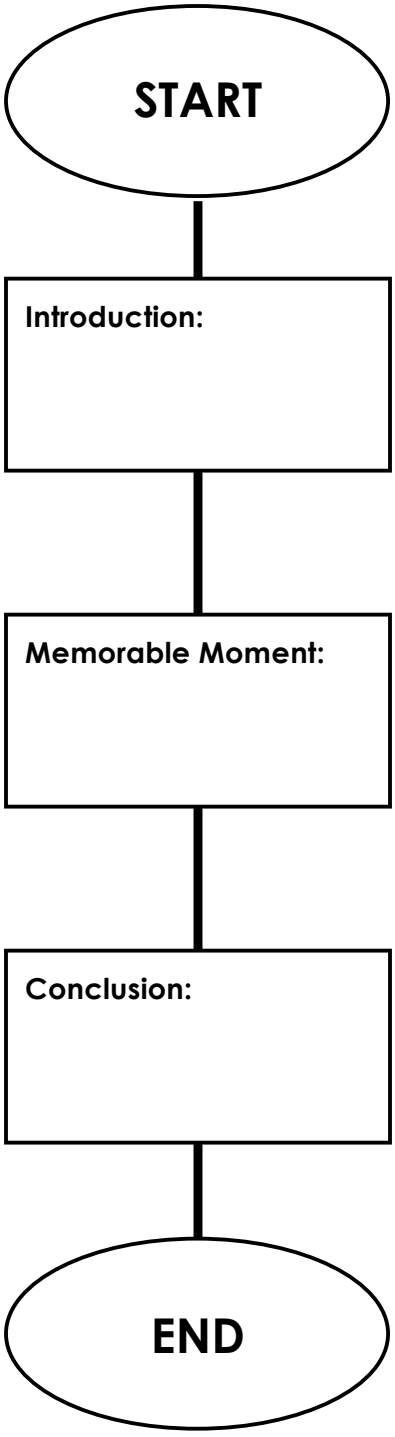
_____

_____

Name/Number: _____ Page #___ of ___

Flowchart for the story about _____.

Step 1: Fill out the Basic Flowchart
(Use your storyboard to help)



Step 2: Add to it!
(Try using more shapes)

Step 3: Keep adding to your flowchart as you learn new programming skills.
(Use a new piece of paper)

12: The Application Cycle

Teacher Lesson Plan

Lesson Rationale-

This lesson elaborates on the application cycle (program, test, evaluate, redesign, etc.) of the Engineering Design process and gives students a framework for how to go about developing their digital story in LaPlaya and how to keep track of their work and elaborate on their program's flowchart(s).

Objectives-

Students will be able to:

- Understand the iterative process of the application cycle within the Engineering Design Process
- Distinguish between each part of the cycle in the context of creating a digital story using LaPlaya (Program, test and evaluate, and redesign)
- Record aspects of the design process
- Understand that they will need to elaborate on their flowchart as they learn more programming skills and create a more complex digital story.

Standards Emphasized-

CSTA Computer Science Standards:
L1:6:CL- Identify ways that teamwork and collaboration can support problem solving and innovation.

Materials-

Teacher	Student
"Engineering Design Thinking" worksheet with projector (optional)	"The Application Cycle" worksheet "Engineering Design Thinking" Worksheet (optional) Piece(s) of paper & pen/pencil for notes and new flowchart iterations Design Notebook (optional)

Learning Tasks and Instructional Strategies-

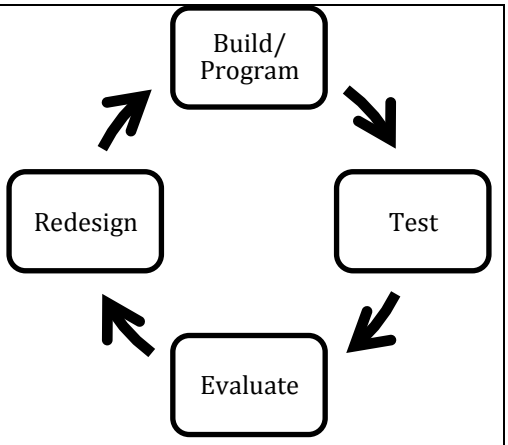
(Total Approx. Time: 10-15 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Go over with the class what the application cycle is (related to scientific process for science, design process for engineering & program development process for computer science), where it fits with the other steps in the Engineering Design process (may want to reference the "Engineering Design Thinking" worksheet), and what each part of it is (Program, Test, Evaluate, and Redesign).	Discussing how this application cycle in engineering design thinking is similar to the scientific process as well as the program development process in computer science is important to allow students to see the relationships between these different fields and the importance of this process overall. Referencing where the application cycle is in the Engineering Design process as a whole is good for allowing students to visualize which part of the process they are currently engaging in. By going through what each part of the design process is in more detail it should help students distinguish between each part and will help them while ultimately programming their digital story.
3-5 min.	Go over with students about how from this point forward students will be using LaPlaya to create their digital story. There is a button titled "Design Thinking Project" that students can click on any time they want to work on their digital story. Let students know the logistics of when they can and will be working on this program during class time (there will be a few dedicated sessions to work on this program at the end of the module and you may consider allowing students to work on this project if they finish the computer-based lessons early).	Giving students the logistics about when and how students will be working on this program is good so they know what they have to do, when they can do it, and what your expectations are for them. Make sure to tell students that they should be using their storyboard and flowcharts to help them in the initial stages of creating their program. They should first decide on the basics including which sprites they will be using for their story and what background(s) they will need and then they can begin adding blocks to program these elements. You should also remind students that they will need to think about the user and the user's experience when using their program.

3-5 min.	Go through the bottom portion of "The Application Cycle" worksheet with students, which gives prompts for the kinds of things that they should be recording throughout the design process and when creating their programs. They can record these things as much as you wish (after every work session, at the end of the module, etc.). Make sure that they understand that they should be going back and creating more complex flowcharts for their program and that they may want to create a new flowchart for each sprite (or the background) that they are using.	Recording problems you encounter, changes you make, and possible solutions to those problems is a huge part of not just engineering, but also computer science and the other sciences. Often times these things are recorded in a science notebook or a design notebook and these are used so you can go back and look at the changes you have made and the process you have gone through, which is good practice in case a problem arises (ex. If a program you write has a glitch you can trace which part of it may be messing up the whole thing). It is also important that students clearly understand that they should be adding to the flowcharts for their program and making them more complex and encompassing of everything they have included.
During work sessions	Throughout the work sessions that students will have to program their digital story you may consider having students think about the following things: • Do I have all the sprites I need? • Do they all do what I want them to do? • Is my background how I want? • Does everything do what it is supposed to when clicked, when a key is pressed, when something happens? • Is there a way I can make my project better, more fun, more user friendly? • Can I use fewer blocks to do the same thing? • What else do I need to add, take away, or change on my flowchart?	These questions should help guide students when programming their digital stories by asking that they think more deeply about the program as a whole, each specific aspect of the program, as well as the different people who may be using their program. This should help them evaluate their own work and create a high quality program.

Name/Number: _____

The Application Cycle

What is the **application cycle?**	The part of the Engineering Design Process where you are actually creating something (by building, making, or programming it). You also test your design and see what could be improved so you can redesign and create a better program.	
What are the parts of the **application cycle?**	**Build/ Program:**	You are creating something based on the design you created. This is where you move to the computer to actually program your digital story.
	Test:	Try out your program. Make sure to think about how others may use your program also.
	Evaluate:	What did and didn't work?
	Redesign:	What should you change about your program?

From this point on you will be working on the computer using LaPlaya to create your digital story! *Keep coming back to your work to keep track of the **constraints** you must follow and draw new, better, and more detailed **flow charts**.*

Keep track of the changes you make throughout the application cycle:

In your Design Notebook or on a separate sheet of paper write the following information down when you make a change to your project:
1. What change(s) did you make to your digital story? *(How is it different from your previous version?)* 2. Why did you make those changes? *(Did it make your project better or worse than you had originally hoped?)* 3. How did you change your flowchart to show the changes you made? *(Did you create new commands/shapes? Did you add more detail? Did you delete part of it? Did you add a new flowchart for a different sprite?)* 4. Is there any way to improve your project? *(Could you make it better? How? What would you add/take away?)*

13: Event-Driven Programming: When Other Sprite Clicked

Teacher Lesson Plan

Lesson Rationale-

In this lesson, students continue creating programs that run on events triggered by a user. In the last two lessons, students programmed the action of a sprite to correspond with a key press or when that same sprite was clicked on the stage. Here, we add one more level of complexity by coordinating action between sprites. In some cases, we may want one sprite to do something based on the action of another sprite. In these tasks, students will program the action of a sprite to occur when a **second sprite** is clicked using the event block, "When Other Sprite Clicked."

Objectives-

Students will be able to:

- Create scripts using the event block, "When Other Sprite Clicked."
- Practice keeping track of multiple sprites doing multiple actions.
- Engage a user in their programs.
- Make a sprite respond to a user clicking a different sprite

Vocabulary-

User: The person playing the game, running the program, etc.
Control Blocks: Blocks that determine when something should happen.
Event: Something that the user does (click sprite, press button, etc.).

New Blocks -

Block	Block Name	Block Description
	When [Other Sprite] Clicked	Event block that runs a script when a second sprite is clicked.

Standards Emphasized-

Materials-

Teacher	Student
Computer Projector (to demonstrate Task 1 to class)	Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*Task 1* Introduction: Intersection 1	Use this task to demonstrate how to create a script with the "When Other Sprite Clicked" event block. Create a script that moves the car to the parking spot. Following the teacher demonstration, allow students to complete the task independently on their computer.	Program the Car sprite to move to the Parking Spot when the Traffic Light sprite is clicked.
	Task 2 Intersections 2	Students program two car sprites to move to their corresponding parking spaces when the corresponding traffic light is clicked. Note: The color of each car matches the house and parking spot.	Program the White car to move to its parking space when the White traffic light is clicked. Program the Blue car to move to its parking space when the Blue traffic light is clicked. Hint: The color of the parking space matches the color of the car!
	Task 3 Intersections 3	Students program two car sprites to move to their parking spaces when the corresponding traffic light is clicked. Each car must turn 90 degrees to the right. Note: The color of each car matches the house and parking spot.	Program the White car to move to its parking space when the White traffic light is clicked. Program the Blue car to move to its parking space when the Blue traffic light is clicked. Hint: Both cars need to turn at the intersection!
	Task 4 Piñata Candy Falling	Students program three candy sprites to move to the ground when the piñata sprite is clicked.	Program the three pieces of candy to glide downwards when you click the piñata. Bonus! Try to make the candies fall so you can see all three of them!
	Task 5 Shaking the Tree	Students program four apple sprites to glide downwards when the tree sprite is clicked.	It's such a windy day! The wind is shaking the fruit from the tree. When the tree sprite is clicked, the tree shakes. Program each apple sprite to glade downwards when the Tree sprite is clicked. Hint: Each apple may need to fall a different distance to reach the ground.

Suggestions for Students who Finish Early-

- Play in the Sandbox!
- Work on Design Thinking digital story

Common Student Errors -

- Students may not realize that they need to program the sprite that will be **doing** the action and not the one that is **triggering** the action (ex: if you want the car to drive when a traffic light is clicked you need to create the script in the **car's** script area, not the traffic light's script area).
- Students may try to reprogram things that are already programmed if they are unlocked or may delete successful scripts that they have already created thinking they are no longer needed.

Teaching Hints -

- Students can copy/paste scripts from one sprite to another (useful in the planets task!). To copy a script, drag the script over to the other sprite in the sprite list (bottom right corner).

Initialization Fired Up

Teacher Lesson Plan

Lesson Rationale-

Computer scientists often want their programs to run in the exact same way every time. In order to do this, items need to be set up properly when the program begins each and every time.

The process of properly setting up conditions for an activity is very common in daily life. For example, whenever you play a game, you must set it up properly when you begin. Before a soccer game starts, all of the players get on their own side of the field, and the ball is placed in the middle. When you play a board game like Monopoly, each player receives the same amount of money to begin. Players do not hold on to hotels or property from game to game!

Initialization is equally important in programming. In traditional programming, initialization involves setting starting values of variables. Initialization in LaPlaya differs from traditional programming in two ways. First, initialization involves placing sprites in the right location, and with the right size and orientation. Second, LaPlaya holds onto values between when you close it and open it again. Only items that changed during the course of the program need to be initialized. Thus, it's in LaPlaya to relate initialization to "resetting" instead of "setting up." Please focus on the wording "setting up."

In this activity, the teacher introduces students to the importance of initializing. The teacher intentionally "forgets" to place items in their starting position for an activity. You can take this as far as you want – doing this in the activity we suggest or throughout the day. We have further suggestions at the end if you want to do it for the entire day.

Objectives-

The purpose of this activity is to reinforce the idea of initialization – setting multiple conditions properly before restarting. There are many ways you may need to set something up, so we encourage you to be creative and find additional ways to reinforce this concept.

Standards Emphasized-

Common Core State Standards:
CCSS. Math.Content.4.G.A.1 - Draw points, lines, line segments, rays, angles (right, acute, obtuse), and perpendicular and parallel lines. Identify these in two-dimensional figures.

Materials-

Teacher	Student
Several erasers An outdoor play area with two parallel lines that are quite far apart	

Learning Tasks -

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks
5 min.	• Begin by dividing the students in two groups. Do not have them stand on the lines the first time. Place the erasers on the ground in the middle of one of the groups. Explain the rules of the game – when you say go, everyone tries to get the erasers. The group who gets the most erasers wins. **If someone raises a point about fairness, move to the next step – do not run the game.** • Run the game once. Presumably, the group in which you placed the erasers will win. Declare them the winners. Someone should raise an issue with the fairness of the game. If not, ask them whether it was fair.
10 min.	• Discuss with the students what was unfair about the game. Identify the problem as not setting up the game properly. • Now, with the same group assignments, have one group line up on one side, and have the other group line up parallel to them with a large gap in the middle. Put the erasers on a third line equidistant from the students. (You may use this as a math lesson – parallel lines, equidistant, etc.) • Play two more times just for fun. ☺
8 min.	• Return to the classroom (either immediately or after more play time) and explain the concept of "setting up" or "initializing" to students. • Invite the students to supply ideas about other times when initialization is required for success. For each one, have students tell you o What needs to be initialized? o What would happen if they forgot to initialize it?

Optional Learning Tasks -

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks
5 min.	These are things the teacher can "forget" to set up properly throughout the day. For example: • When you write on the chalkboard, forget to erase it before using it again. • Xerox a worksheet on top of another worksheet. You can do it once and then use that image to Xerox the rest.

14: Initializing Programs

Teacher Lesson Plan

Lesson Rationale-

Computer scientists often want their programs to run in the exact same way every time. In order to do this, any items that may have changed over the course of the program need to be reset to the starting conditions before the program runs again. This is initialization. In LaPlaya, sprites can move, change size or color, or point in different directions during a program. In order to initialize, students need to reset these attributes to their starting conditions without a user being able to notice (initialization happens in the background). If a program is initialized correctly, the user will not see how the sprites changed, only that they did change. Sprites will instantly appear in their correct location with the starting size and orientation! In these tasks, students begin to understand the importance of initializing in computer science by playing games that were not reset correctly. Then, students will learn how to initialize in LaPlaya by creating special scripts with the event block, "When Get Ready Button Pressed".

Objectives-

Students will be able to:

- Identify attributes of a sprite that need to reset when a program runs, such as position, size, color and orientation.
- Create a program that moves a sprite immediately to the starting position with the Go To blocks.
- Create initialization scripts using the event block, "On Get Ready Button clicked"

Vocabulary-

Initialize: To set or reset any changes in a program to the starting values.
Starting Conditions: The many ways a sprite should begin when a program is initialized (color, location, orientation, etc.).
Orientation: The direction something is pointed

New Blocks -

Block	Block Name	Block Description
	When Get Ready Button Clicked	Event block that runs a script when the Get Ready button is pressed. Used to create an initialization script.

Standards Emphasized-

Materials-

Teacher Computer Projector (to demonstrate Task 1 to class)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
	Task 1 Demo: Watch the Tortoise and the Hare	Students watch a video in LaPlaya of Tortoise and the Hare. Video begins after pressing the "Get Ready" button followed by the "Green Flag" button.	Click the "Get Ready" button and then the "Green Flag" button. Watch the story of the Tortoise and the Hare.
	Task 2 Demo: Tortoise and the Hare Version 2	Students will learn the importance of initialization through a common childhood story. Part 2 involves a fully functioning story for the students to watch after pressing the "Get Ready" button followed by the "Green Flag" button. However, when the student watches the story a second time everything goes wrong because nothing has been initialized.	Watch the story of The Tortoise and the Hare once. Then watch it one more time and explain the difference between the two.
	Task 3 Piñata Part 2	Students create initialization scripts to reset a piñata. They program each candy sprite to start in the piñata.	Oh, no! Someone forgot to start the program with the candies in the piñata! The kids will be so disappointed! Program the "get ready" scripts for the candies so that they start in the piñata. The first candy has a picture of the script so you can see how to do it. Program the script in all three candies.
	Task 4 Animal Sprint: Horse	Students program the "get ready" initialization script that will make a horse sprite start in the proper place for a sprint. The position is marked by a sprite that is the outline of the horse.	Oh, no! The horse is not at the starting line for the race! Program the "get ready" initialization script that will make the horse start in the proper place (marked by a sprite that is the outline of the horse).

	Task 5 Animal Sprint: Cat	Students program the "get ready" initialization script that will make a cat sprite start in the proper place with the correct size for a sprint. The position and size are marked by a sprite that is the outline of the cat.	Oh, no! The cat is not ready for the race! Program the "get ready" initialization script for the cat. Then run the program (press the "get ready" button, then the"green flag" button). What changed about the cat? Make sure your "get ready" initialization script initializes everything about the cat.
	Task 6 Animal Sprint: Rooster	Students program the "get ready" initialization script that will make a rooster sprite start in the proper place with the correct orientation for a sprint. The rooster needs to point left to start. The position and orientation are marked by a sprite that is the outline of the rooster. Students may need to run the program more than once to realize they need to initialize the orientation of the rooster.	Oh, no, what is that rooster doing! It needs to get ready for the race! Program the "get ready" initialization script for the rooster. Then run the program and watch the rooster. Make sure your "get ready" script initialized everything about the rooster.

Suggestions for Students who Finish Early-

- Play in the Sandbox!
- Work on the Design Thinking digital story.

Common Student Errors -

- Students need to distinguish between the scripts for the "green flag" button and the "get ready" button because the "get ready" button is used to **initialize** the program and the "green flag" button is used to **start** the program.
- Students might not initialize ALL aspects of the sprites that need to be (ex. Position tends to be easier for them to notice and initialize, but position and orientation tends to be left undone.

Teaching Hints -

Programs need to be initialized so every time you open them it starts in the appropriate way. You may want to use the example of when you open a game on a phone. It opens to a screen or a menu that is always the same regardless of where you finished last time. Initialization is hidden from the user and is different from the "start" of the program.

15: Animating Sprites

Teacher Lesson Plan

Lesson Rationale-

Animations are important for telling a digital story. Often, digital writers want to do more than place images on screen, they want images to move. In LaPlaya, animations are created when images of a sprite change in small ways much like a flipbook. On each page of a flipbook, animators draw pictures that chang slightly from page to page. When these pages are turned rapidly, the images on the pictures seem to move. And the faster someone turns the pages, the faster the images seem to move! (See a YouTube flipbook of the 2014 World Cup as an example).

Unlike flipbooks though, computers can alternate through each image so quickly that users will not see the animation (like skipping from the first straight to the last picture in a flipbook). Because of this, computer scientists write special programs for **timing**, code that pauses each image briefly.

In LaPlaya, animating works by rapidly moving through images of a sprite called **costumes**. Timing is necessary to determine what kind of visual effects are created! When there are long waits between costumes, the animation seems slow; when the waits are short, the animation seems fast. In this activity, students will practice animating sprites by programming a series of costume changes using wait blocks.

Objectives-

Students will be able to:

- Animate a sprite through costume changes.
- Practice implementing initialization scripts.
- Properly time an animation with waits blocks between costume changes.
- Create new costumes for a sprite.

Vocabulary-

Costume: A picture that represents a sprite or a version of that sprite.
Costume Change: The process of moving between pictures (costumes) of sprites in LaPlaya
Animation: Using multiple and successive drawings of a character to create the illusion of movement.
Timing: Choice of when something should occur.

New Blocks -

Block	Block Name	Block Description
	Switch to costume _____	Look block that changes the appearance of a sprite to another costume that is selected.
	Wait __ secs	Control block to add a variable

		amount of time between actions.
	Next Costume	Optional: Look block that changes the appearance of a sprite to subsequent costume on the list.

Standards Emphasized-

Materials-

Teacher	Student
Computer Projector (to demonstrate Task 1 to class)	Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*Task 1* Ballerina	Teacher demonstrates how to move between costumes for a single sprite with the example of a dancing ballerina. Students then complete the task independently on their computer.	If you just make the sprite glide, it does not look like it's really moving. Using costumes, we can make the sprites do many fun actions! Program two scripts for the ballerina. 1) Initialize the ballerina's position and starting costume. 2) Make the ballerina dance when she is clicked.
	Task 2 Pick a Dancer	Students create scripts to animate dancer sprites. They may choose from three dancers to program, and need to use at least three costume changes in the routine.	Now make a dance routine yourself! First pick a dancer to program. Then write a script for a dance routine that includes 3 costume changes. Hint: Vary the wait blocks to make the dance slow or fast!
	Task 3 Dance Party!	Students create scripts to animate a dance party. They program each dancer sprite to animate when the disco ball is clicked.	Now, create a dance party! Program each dancer to animate when the disco ball sprite is clicked. Hint: Start with the break-dancer!
	Task 4 Draw Your Own	Through a tutorial, students learn how to create their own costumes and animate a stick figure with	If you can't find a sprite you like, then you can draw your own! Click the "get ready"

		those new costumes.	button, then the green flag button to go through a lesson on how to draw your own sprite costume. Click on each Next button on the stage to learn steps to drawing your own costumes for Stan the Stick Figure.

Suggestions for Students who Finish Early-

- Play in the Sandbox!
- Work on the Design Thinking digital story.

Common Student Errors -

Students tend to forget to add the "wait" block between costume changes and can get frustrated when they don't see all their costume changes that they programmed. In reality, they just happen too fast to see, which is why the wait block is necessary.

Teaching Hints -

- Bring a flipbook to class to emphasize animation.
- Allow students free time during class to create their own flipbooks with paper and pencils.
- Encourage students to use different times in the wait blocks for their scripts. Not every costume change needs to the same timing – wait blocks can differ in time for each action!

16: Changing Scenes

Teacher Lesson Plan

Lesson Rationale-

In this project, students learn how to change scenes in LaPlaya. Often, in a program or game, computer scientists want different aspects of their projects visible at different times. In a game, the scene may change when the player travels to a different location or moves to a higher level. Or, in a digital story, a scene will change as characters travel to new places and meet new characters. In LaPlaya, scene changes require two things 1) Figuring out which sprites are visible, 2) Changing the stage background. In this activity, students practice moving between scenes by determining when the change is going to happen, what the new background will be, and which sprites will be visible in each scene.

Objectives-

Students will be able to:

- Use show/hide blocks to determine which sprites are visible in each scene.
- Change the stage background.
- Change to new backgrounds with different events.

Vocabulary-

Scene: A set of sprites and backgrounds that serves as setting to a program.
Background: Representations of the stage and part of the scene.
Show: Make a sprite appear or reappear on the stage.
Hide: Make a sprite disappear from the stage.

New Blocks -

Block	Block Name	Block Description
	Hide	Look block that makes a sprite immediately disappear.
	Show	Look block that makes a sprite immediately appear or reappear.
	Switch to background __	Look block that changes the background of the stage to a selected background.
	Next background	Optional: Look block that changes the appearance of the stage to the following background on the list.

Standards Emphasized-

Materials-

<u>Teacher</u> Computer Projector (to demonstrate Task 1 to class)	<u>Student</u> Computer Storyboarding worksheets (2 pages) Pencil/Pen

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	**Learning Tasks**	**Purpose**	**Student Directions**
5 min.	*Task 1* Plant Growing Demo	Teacher demonstrates the complete Plant Growing program, which contains three scenes. Click on the different number sprites to change scenes. Students then complete a storyboard of the three scenes by determining: 1) Visible sprites in each scene. 2) Background used in each scene. 3) Sprites who do something special when clicked. During the next tasks, students will incrementally write scripts for each of these changes.	Let's storyboard this program! Using the worksheet, draw the three scenes in this program. Be sure to draw: 1) Sprites you can see. 2) Where the sprites are in each picture. 3) If a sprite does something special. 4) The background.
	Task 2 Plant Growing Initialization	Students create initialization scripts for all sprites in Plant Growing. These are the starting conditions for each scene. Students need to initialize the location, costume, and visibility (hide/show) for each sprite. Then they need to initialize the starting background.	Look at the storyboard you made for the plant growing project. Write scripts to initialize all of the sprites and the background. Remember to initialize the costume, location, and whether it is visible. The code for the plant has been given to you.
	Task 3 Plant Growing	Now, students program scripts to change from scene 1 to 2.	Write the scripts to change from Scene 1 to Scene

	Scene 2	Clicking script 1 and 2 on the stage triggers the scene change. Students look at their storyboards to determine how each sprite and background changes in scene 1 and scene 2. They need to show/hide different sprites, pick correct costumes for each sprite in the two scenes, and position the sprites in the correct place.	2. When the 2 sprite is pressed, the scene should change to scene 2. When the 1 sprite is pressed, the scene will change to scene 1. *Hint: Scripts for the plant have been started for you, and the initialization scripts are already complete!*
	Task 4 Plant Growing Scene 3	Students complete the program by creating scripts to change between Scene 2 and Scene 3.	Now you are adding scripts to change to Scene 3. Use everything you learned in the last two tasks and look at your storyboard!

Suggestions for Students who Finish Early-

- Play in the Sandbox!
- Work on the Design Thinking digital story.

Common Student Errors -

Teaching Hints -

- Use the storyboards from the earlier design-thinking activity to connect how scenes changes work and how to use the "plant growing storyboard" for this lesson.
- Because there are a lot of things going on to create scene changes students can use storyboards (included in students handouts for this lesson) to keep track of what everything is doing, what it looks like during each scene, and what needs to change (background & costume changes).

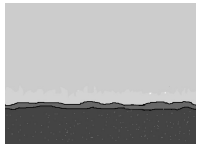
Name/Number: _____

PLANT GROWING STORYBOARD

Create a drawing for each scene.

Scene 1	Scene 2	Scene 3
Description: _____ _____ _____ _____ _____	Description: _____ _____ _____ _____ _____	Description: _____ _____ _____ _____ _____

Circle which version of the sprite or background is correct for each scene.

		Scene 1	**Scene 2**	**Scene 3**
Stage		Scene 1 Scene 2 Scene 3	Scene 1 Scene 2 Scene 3	Scene 1 Scene 2 Scene 3
Plant		Show Hide Seed Sapling Flower	Show Hide Seed Sapling Flower	Show Hide Seed Sapling Flower
Sun		Show Hide ClickMe ShiningSun RegularSun	Show Hide ClickMe ShiningSun RegularSun	Show Hide ClickMe ShiningSun RegularSun
Cloud		Show Hide Raining Not Raining	Show Hide Raining Not Raining	Show Hide Raining Not Raining
Three		Show Hide Selected Normal	Show Hide Selected Normal	Show Hide Selected Normal
Two		Show Hide Selected Normal	Show Hide Selected Normal	Show Hide Selected Normal
One		Show Hide Selected Normal	Show Hid Selected Normal	Show Hide Selected Normal

17: Sharing Your Work

Teacher Lesson Plan

Lesson Rationale-

After students have spent the allotted class time working on their digital stories they will finish them up and get them ready to share with others. This lesson underscores the importance of sharing your work in not just engineering, but also the sciences, programming, design, etc. This also introduces students to the idea of providing others with constructive criticism to help them create a better program.

Objectives-

Students will be able to:

- Give constructive feedback on each other's work
- Share their digital story with another person or people.

Standards Emphasized-

Materials-

Teacher	Student
**Determine who/where your students will be sharing their digital story programs with and whether or not they will be provided with feedback on their work.	"Engineering Design Thinking" worksheet (optional) Design Notebook (optional) Computer (to share their program)

Learning Tasks-

(Total Approx. Time: 10-15 minutes)

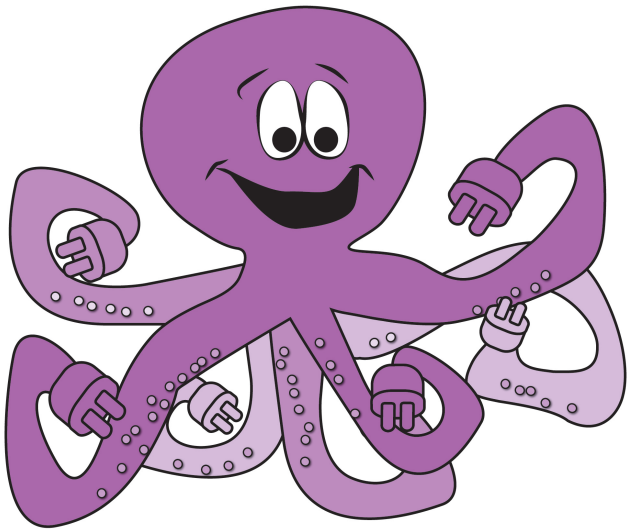
Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 mins.	Introduce the last part of the Engineering Design process; sharing your work. Discuss the importance of sharing your work with others and how by doing so you can get helpful feedback to help improve your program. Explain that sharing your work is done in many fields- the sciences, engineering, programming, design- and is hugely important in creating the best possible products, processes, or ideas.	It is important to emphasize that the design process is a cycle and once you share your work and get feedback on it you may wish to go back to the beginning and create a new design that is even better. It is important that students understand that feedback and even criticism from others can be a good thing that motivates them to make their program better.

3-5 mins.	Discuss with the class the importance of giving constructive criticism when looking at others' work. This means that you are not only making remarks about their work, but you are offering suggestions for how they could improve their program and also including things that they did well. Also explain that just because your project isn't perfect and someone gives you feedback they are not trying to be mean, but are trying to help you.	By teaching students how to give and receive constructive criticism they can help each other develop better ideas that multiple people have worked on. This collaborative working atmosphere is crucial for many fields and will be a critical skill for students to be familiar with in life and school (playing a team sport, being in a club, working on a group project, etc.)
3-5 mins.	Have students share their work online. You may decide that students simply save their project on the LaPlaya website (in that case students are sharing their work with us), or you may chose to have them share their work with others, or somewhere online. This choice is up to you. Options for sharing students' work—teacher only, other students in the class, another class, parents, or an online community.	It is important to make it clear to students who will be seeing their work and whether or not they will be receiving feedback from those people (we will not be providing feedback, but you could ask that their classmates or students in another class do). As an optional extension, if students are provided with feedback from others they may go back to a previous step in the design process (either design—storyboarding and flowcharts—or the application cycle) and make improvements to their program based on that feedback. In this case the iterative nature of the Engineering Design process is truly highlighted.

KELP-CS

Curriculum

2015-2016



THE UNIVERSITY OF CHICAGO

CEMSE
CENTER FOR ELEMENTARY MATHEMATICS AND SCIENCE EDUCATION
THE UNIVERSITY OF CHICAGO

Diana Franklin
Computer Science Education
dmfranklin@uchicago.edu

UCSB
UNIVERSITY OF CALIFORNIA
SANTA BARBARA

The Gevirtz School
Graduate School of Education

Danielle Harlow
Science Education
dharlow@education.ucsb.edu

Acknowledgements

Project Directors

Diana Franklin, Center for Elementary Mathematics and Science Education
University of Chicago

Danielle Harlow, Department of Education
UC-Santa Barbara

Development Staff

Hilary Dwyer
Johan Henkens
Charlotte Hill
Ashley Iveland
Alexandria Killian

James Cheng-yuan Hong
Sharon Levy
Timothy Martinez
Iris-Eleni Moridis
Logan Ortega
Kenyon Prater
Jenny So
John Thomason
Rick Waltman

Elementary Teacher Consultants

Larry Kelman
Tracey Schifferns
Janis Spracher

Pilot Test Teachers

Bridget Berg-Gankas
Mary Lou Furrer
Cati Gill
Shauna Hawes
Ashleigh Lemp
Jamie Thompkins
Laurie Thorbjornsen

This project is supported in part by the National Science Foundation Grant #1240985.
Additional support provided by the University of California-Santa Barbara

Table of Contents

Introduction

Welcome to Kids Engaged in Learning Programming (KELP-CS)! KELP-CS is a modular curriculum for 4th-6th graders created by an interdisciplinary research team at UC Santa Barbara. Each of our modules consists of 13-14 hours of instruction, and can be used for each grade level. The first module introduces students to block-based programming and digital storytelling. The second develops these programming skills further and teaches students about game design. Moreover, KELP-CS emphasizes design thinking, the process by which engineers develop innovative solutions to problems. Design thinking is a core part of new science standards for K-12 students, the Next Generation Science Standards (NGSS) and involves understanding the problem, generating ideas, selecting an idea based on multiple constraints, and improving the idea. We see design thinking overlapping with computer science in ways that allow students to access each in new and exciting ways!

Each KELP-CS module consists of activities that are completed either in the classroom or on a computer. During classroom activities, students discuss, collaborate, and engage with computer science without the computer. These off-computer activities introduce and reinforce concepts that students need to apply on the computer. On the computer, students complete small, discrete programming tasks in our interface to learn about a larger computational thinking idea such as sequential motion, event driven programming, and user interaction. As students finish these computer activities, they will transfer these programming skills directly to their design-projects. These design-projects span each module to provide students opportunities to iteratively revise and improve their programs.

For our programming environment, we use a block-based programming environment, called LaPlaya that runs through any Internet browser. The interface is user friendly and age-appropriate upper elementary school students. Instead of programming by typing individual lines of codes, students can snap individual command blocks program. As students progress through our modules, more and more blocks are introduced and at any time they can use our Sandbox area to explore the entire language.

We believe that increasing the opportunities for elementary school children to learn computer science is an essential aspect of preparing students for computer science careers as well as preparing all students to be comfortable and adept with technology. Our goal is to create a curriculum that any elementary school teacher can implement with their class during an academic day. We invite you to explore computer science with us, and help prepare the next generation of computer scientists.

Best wishes

The KELP-CS Design Team

KELP-CS: Scope and Sequence

Module 1: Digital Storytelling

Module 1 is intended as a 4th, 5th, or 6th grader's first introduction to computational thinking and programming. Students learn general programming concepts in the context of a particular language and programming environment (LaPlaya). Some general concepts are the importance of placing things in order in a program, breaking down complex tasks into their basic components, and properly associating code with the events that trigger them, and managing the complexity of several sprites moving at once. LaPlaya programming concepts they will learn are sequential programming, event driven programming, costume changes, and scene changes. As students learn and practice these concepts, they will apply them to their animated story, satisfying design thinking standards.

Module 2: Game Design

Module 2 is geared towards 5th grade students who have completed Module 1 (Digital Storytelling) and are already familiar with basic computational thinking and programming with our block-based, programming environment (LaPlaya). This module teaches students about loops, decision-making, message passing, variables, and the game design process. Further, students will learn more complex programming by implementing these ideas in LaPlaya. Through the module, students will learn about and develop skills using the engineering design process. As a culminating project, they will apply all they have learned to create a game.

Introduction to Computer Science

Overview for Teachers

What is Computer Science?

There is often confusion over the swirl of terms related to computing, technology, and computer science that can be daunting when trying to make decisions about what students need to learn related to computer science. Computer science, as defined by the Computer Science Teachers Association (CSTA) and the Association for Computing Machinery (ACM), is "An academic discipline that encompasses the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society".

Computer science is not just about the use of computers or computer applications, it also includes the knowledge and skills necessary to build the next generations of software and hardware tools that the world needs. The national urgency to improve science, technology, engineering, and mathematics (STEM) education is palpable, as officials have called for reforms in these areas, but what is not clear to policy makers at all levels is that computer science is frequently left out of these initiatives. Computer science drives innovation in all STEM disciplines but it is also a distinct discipline with an extensive body of knowledge.

Computer science teaching sits on a continuum from basic computing concepts that can be attained at elementary and middle school levels to deeper knowledge, skills, and practices more appropriate for secondary school. At the elementary school level students should be introduced to foundational concepts in computer science by integrating basic skills in technology with simple ideas about computational and algorithmic thinking. Learning standards developed by an ACM task force (*A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum committee*) include the following for grades three through five that are included in the KELP-CS curriculum:

- Be comfortable using keyboards and other input and output devices.
- Discuss common uses of technology in daily life and the advantages and disadvantages those uses provide.
- Use technology tools for individual and collaborative writing, communication, and publishing activities to create presentations for audiences inside and outside the classroom.
- Use online resources to participate in collaborative problem-solving activities for the purpose of developing solutions or products for audiences inside and outside the classroom.
- Use technology resources for problem-solving, self-directed learning, and extended learning activities.

Why teach Computer Science?

No other subject will open as many doors in the 21st Century, regardless of a student's ultimate field of study or occupation, as computer science. At a time when computing is driving job growth and new scientific discoveries are happening all the time, gaining a deeper

knowledge of computer science and its fundamental aspects is essential not only to have a clear understanding of "what is going on under the hood" of computer software or hardware, but also to develop critical thinking skills that will serve a student throughout his or her career. The U.S. Bureau of Labor Statistics projects that the computing sector will have 1.5 million job openings over the next 10 years, making this one of the fastest growing economic fields

The knowledge and skills imparted by computer science also enables innovation and opens doors. Many fields of science and business depend on computer science. Despite the incredible diversity of the U.S. workforce, it is clear that most of today's jobs depend on some knowledge of, and skills to use computing technologies. It is also clear that this trend is growing as computing becomes embedded more deeply in everyday commerce and society. Computing touches everyone's daily lives; Securing our cyber-infrastructure, voting in elections, protecting national security, and making our energy infrastructure more efficient are among numerous issues dependent on computing and a strong computing-savvy workforce. If K–12 schools are seeking to make students college- and career-ready, computer science should be part of the core curriculum.

Increasing the opportunities for elementary school children (especially girls and other underrepresented minority groups) to learn computer science is an essential aspect of preparing students for computer science careers as well as preparing all students to be comfortable and adept with technology. Research has shown that students' career aspirations at eighth grade are strong predictors of whether they will attend college and whether they will pursue careers in science or engineering, highlighting the importance of experiences children have prior to eighth grade. Fortunately, more children are gaining experiences in computer science at younger ages. Programming environments designed for children and novices (e.g., Scratch, Alice) are easily accessible by classroom teachers and coding in K-12 education has become a huge movement.

What is unclear to many educators is what curriculum will support this growing trend. Although we are beginning to understand how best to teach computer science at the high school level and middle school level, we know comparatively little about effective instruction at the K-5 level. Luckily, some systematic curricula, such as KELP-CS, have been developed that are targeted at helping elementary school students learn programming and computer science. This kind of curriculum is based upon the higher tiers of Bloom's cognitive taxonomy (involving design, creativity, problem solving, analyzing possible solutions to a problem, collaboration, and presenting work) and develops and extends logical thinking and problem-solving skills that can be applied to real world problems.

How will your students be learning Computer Science?

In addition to engaging in the computation thinking and problem-solving aspects of computer science, students will also be introduced to and gain a foundation in an important aspect of computer science; **programming**. Using the modified Scratch environment, called LaPlaya, to either debug existing programs or create new programs students will learn important computational thinking and programming skills including:

- Sequencing
- Breaking down actions

- Event driven programming
- Initialization
- Animation
- Scene changes

Using the KELP-CS curriculum students will also be engaging in many computer science practices such as precision, optimization, utilizing tools strategically, and switching back and forth between thinking as a software developer and thinking like a software user.

References

Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough: the educational theory and research foundation of the exploring computer science professional development model. In *Proceedings of the 45th Technical Symposium on Computer Science Education (SIGCSE '14)*. Atlanta, GA: ACM.

Liberman, N., Kolikant, Y., & Beeri, C. (2012). "Regressed experts" as a new state in teachers' professional development: lessons from computer science teachers' adjustments to substantial changes in the curriculum. *Computer Science Education*, *22*(3), 257-283. doi:10.1080/08993408.2012.721663

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cuniff, D., ... & Verno, A. (2011). CSTA K-12 Computer Science Standards. *CSTA Standards Task Force*.

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). Running on empty: The failure to teach K-12 computer science in the digital age. Association for Computing Machinery. *Computer Science Teachers Association.*

Zendler, A., & Hubwieser, P. (2013). The influence of (research-based) teacher training programs on evaluations of central computer science concepts. *Teaching & Teacher Education*, *34*130-142. doi:10.1016/j.tate.2013.03.005

Introduction to Engineering Design Thinking

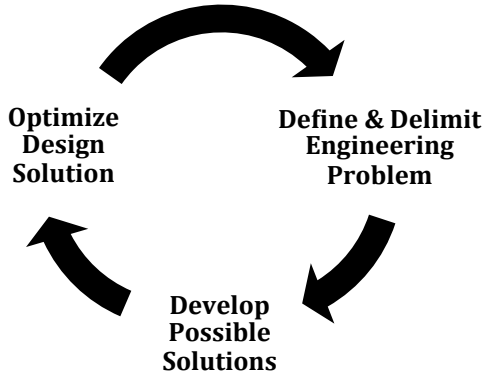
Overview for Teachers

What is Engineering Design Thinking?

Design thinking is the process by which engineers develop innovative solutions to problems and is now a core idea in new science standards for K-12 students, the Next Generation Science Standards (NGSS). The process involves understanding the problem, generating ideas, selecting an idea based on multiple constraints, and improving the idea. Even before children enter school, they use the process of design thinking to help them solve problems.

Consider a child who lets go of the string of a helium-filled balloon in her kitchen. The balloon, now on the kitchen ceiling becomes too high for her to reach, presenting a problem to solve – How to reach the balloon? She may consider the many resources available to her right in her kitchen such as chairs and cooking utensils. First, she might try using a chair to stand on. If the chair is not high enough, she might try to hold a wooden spoon in her hand to extend her reach while standing on the chair. If she then notices that she can hit the string, but not grasp it with the spoon, she stands on the chair holding a set of tongs, a solution that would allow her to reach and grasp the string. We encounter these sorts of engineering problems and engage in this sort of problem solving or design thinking every day.

The Next Generation Science Standards (NGSS) breaks design thinking into three stages: 1) Defining and delimiting an Engineering Problem, 2) Developing Possible Solutions, and 3) Optimizing the Design Solution The table below describes what students in grades 3-5 should be able to do to demonstrate understanding of these three stages.



Define/Delimit Problem.	Define a simple design problem reflecting a need or a want that includes specified criteria for success and constraints on materials, time, or cost.
Develop solutions.	Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.
Optimize design solution	Plan and carry out fair tests in which variables are controlled and failure points are considered to identify aspects of a model or prototype that can be improved.

The first step is to **define the problem**. An engineering problem includes **goals** (or **criteria for success)** for what should be done (e.g., build a bridge that spans a stream that is 5 meters wide and can support the weight of 5 adults) and **constraints** (e.g., a specified budget, limit on materials or time limits). The first step to solving any engineering problem is to fully understand the problem including the goals and constraints.

The second step is to **develop solutions**. This step includes brainstorming or generating several different ideas and then considering how well each idea is likely to meet the problem goals and staying within constraints.

The third step is to **optimize the solution.** Once an engineer has selected a solution, the next step is to optimize that solution. Optimize means to take an idea and make that idea as good as possible. In engineering, there is usually not a perfect solution and multiple factors are involved. Sometimes as one tries to increase optimization along one factor (e.g., using less material), it may mean decreasing performance along another (e.g., speed).

How is Engineering Design Thinking related to Computer Science & Programming?

Engineering Design Thinking is an important part of computer science and programming. When programmers create a game or an app or any other piece of software, they engage in design thinking. They consider the problem they are solving (e.g., how can I make a fun game for 4th graders to learn about math?) and then develop and optimize their solutions.

How will your students be engaging in Engineering Design Thinking?

In this design thinking activities that complement the programming activities in KELP-CS, children will be developing their own piece of software (a "Digital Story" in Module 1). They will learn tools that are useful to computer scientists like storyboards and flow charts as they develop and optimize their digital stories.

References

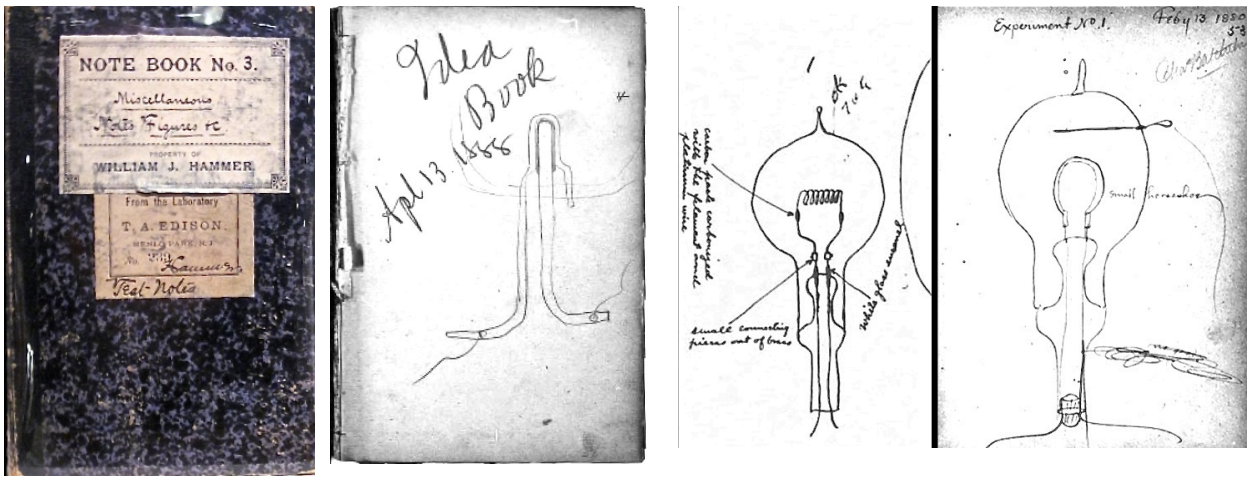
National Research Council. (2012). *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. Washington, DC: The National Academies Press.

Achieve (2012). *Next Generation Science Standards.* Available at http://www.nextgenscience.org/

Using Design Notebooks

Why use a Design Notebook?

The idea of using a notebook to organize thoughts and ideas and keep track of work is not a new idea. Inspiring people throughout history have used notebooks like these to jot down ideas for books or movies (like Mark Twain and George Lucas), record observations of nature (as Thomas Jefferson and Charles Darwin did), keep track of business (like John D. Rockefeller), or flesh out scientific and engineering ideas and designs (as Isaac Newton and Thomas Edison did). Creating a design notebook is now a common practice in engineering, the sciences, and product design. This is a place where individuals or collectives can throw ideas out, incubate them, come back to them, and most importantly, develop them to fruition.



Thomas Edison's Design Notebook with initial sketches of the incandescent bulb.
Photos courtesy of Edison.rutgets.edu

How will your students be using a Design Notebook?

If you chose to have your students utilize a design notebook throughout the KELP-CS curriculum, students will be using either a composition notebook or spiral bound notebook to keep all of their worksheets, notes, and designs together for the "Design Thinking" project that they will be working on throughout each module.

This design notebook should include a title page with information such as student login information (usernames/passwords) and may include a table of contents (if you are planning on going through their design notebooks this is a good way to navigate them more easily).

Throughout the design thinking lessons students will have worksheets that they can cut and paste into the pages of their design notebooks and can then take notes on the subsequent pages, ensuring that everything is in order and easy to find. Once students begin programming using LaPlaya they will be keeping notes on what they did on the computer,

what did/didn't work, and what they changed in their program. They will also be creating basic flowcharts for their program (an essential software development skill and a required course in undergraduate computer science) and will be adding to them as they learn new, complex commands and adapt their digital stories.

A design notebook is ideal for the KELP-CS curriculum because students will be going back and forth between online and offline activities and will not always be working on their Design Thinking project (especially when you consider that they may not even work on this curriculum each day) so by having everything in one place that is easily accessible and in order students can more easily get back into the proper mindset to work on their project and can easily reference previous material if they desire.

The scope of how your students use design notebooks and engage in the design thinking activities is entirely up to the individual teacher and depends on the individual class. You may consider expanding on some of these lessons to incorporate more writing on or off the computer that would allow you to cover multiple standards at once (Common Core Writing as well as Next Generation Science Standards Engineering Design Thinking). You may also want to include more collaboration in these lessons and have students share their work with each other and can write feedback and suggestions in the margins or on the back of the page. This may also be a valuable tool for you to provide feedback to your students or should you choose to evaluate their work.

References

McKay, B., & McKay, K. (2014, January 1). The Pocket Notebooks of 20 Famous Men. Retrieved July 30, 2014.

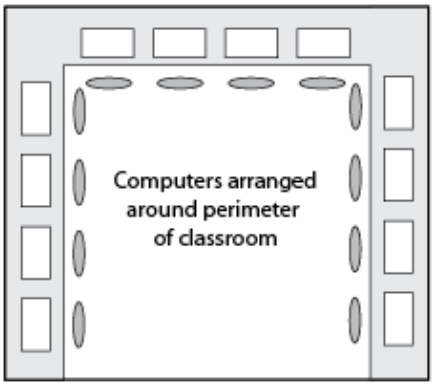
Tips for Teachers

With the help of our pilot teachers and teacher consultants we have compiled a list of tips to assist with common issues that take place in the context of teaching a computer science curriculum in the elementary school classroom such as KELP-CS. We discuss common issues that arise based on the configuration of the learning environment, the mindset of teachers and their students, the grouping of students, and the logistics of working on the computer as well as tips that may help with these issues should they arise in your classroom.

Classroom/Computer Lab Configuration

There are some basic configurations that many classrooms and computer labs fall into and with each configuration comes certain challenges to teaching. Below we will go through some of the possible configurations and common issues that teachers may run into while teaching the KELP-CS curriculum in each.

Around the Perimeter:



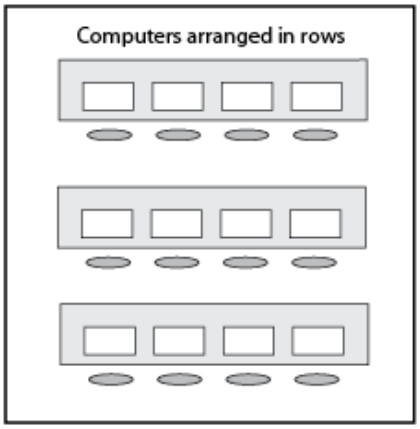
Potential Problems-

When all students face the walls when they are working on their computer it may be difficult for the teacher to get their complete attention when giving instructions are the center of the classroom.

Possible Solutions-

You might consider an easy solution; have the students turn their chairs all the way around to face the center of the room or have the students get up and sit on the floor at the center of the room. Another idea to get students to stop working on the computer and get their attention is to have them put their hands in the air (a big computer stretch) and then turn to the center.

Rows:



Potential Problems-

Because the teacher is usually at the front of the room and cannot see the students computer screens it is difficult to ensure that students are paying attention to the teacher and not distracted with their computer.

Possible Solutions-

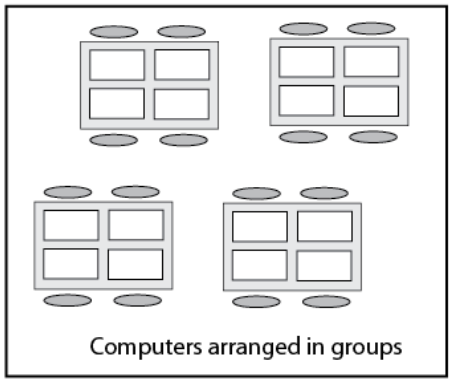
There are several potential solutions for this configuration including having the teacher give directions at the back of the room and having students turn around (like in the "around

the perimeter" configuration). You can also ensure that students' screens don't distract them by having them turn the monitor sideways (if possible), having them turn the monitor off (which shouldn't affect their work only turns off the display), or use a simple piece of paper (shown below) to cover the screen when needed and can be flipped up when students are working on the computer. A piece of paper taped to the top of the computer screen is a great way to ensure that when you're talking students cannot look at their computers. And you can easily see if any student doesn't have their screen covered because it flips to the back where you can see it from the front of the room.

Groups:

Potential Problems-

Like in the "rows" configuration, it is difficult for the teacher to see all students' computer screens at one time and students may be distracted when they should be listen to directions. Another potential problem that may arise is that students will work together more frequently (which is a good thing), but some students may feel less comfortable working with some other students and may take a back seat and let another students do all the work.



Possible Solutions-

If using personal computers, only allow students to have their computers on their desk when they will be using them. You may even ask students to shut their computers or put them away all together if the class is having difficulty paying attention to instructions. To help with group work you may want to circulate around the room while students are working on the computer and make sure no students is being left out. You may also consider setting some guidelines for students ensuring that all students work on their own computer and must complete their own work unless explicitly stated they work in groups.

Individual Desks:

Potential Problems-

Sometimes students get antsy and move around a lot in their desk. If they are using a personal computer at their individual desk and it is not stable there is a chance that the computer could fall off their desk (which may damage or break it).

Possible Solutions-

You can just warn students about moving their desk too much because the computer might fall or you may consider using a different classroom configuration with multiple desks put together, which would stabilize them and prevent the desks from moving around much.

"Fixed" vs. "Growth" Mindset

Research has shown that there is a continuum of beliefs about where success comes from. The ends of this continuum are called "Fixed Mindset" and "Growth Mindset."
Students with a fixed mindset believe that success comes from innate abilities, while those with a growth mindset believe that success comes from hard work and training. Students with a fixed mindset are likely to fear failure and are less likely to try things that they do not already know they will be successful with. In contrast, students with a growth mindset are likely to continue to work at tasks that they find challenging.

Fixed Mindset	**Growth Mindset**
• Intelligence is innate and does not change	• Intelligence can be developed
• Desires to look smart	• Desires to learn more
• Avoids challenges	• Embraces challenges
• Gives up easily when faced with obstacles	• Persists when faced with obstacles
• Worse performance	• Better performance

Students are likely to come into computer science activities with a wide range of experiences. Students with fixed mindsets may see other students (who have more experience) being successful and determine that they are "not good at computer programming." It is important to encourage a **growth mindset** and remind students that, through experience, they will get better.

Encouraging your students to have a growth mindset
Praise effort, "I see you worked really hard on that task"
Help students see that there are things they can learn even when they are unsuccessful.

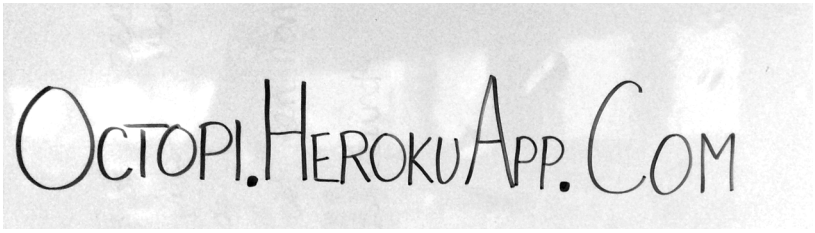
For more information, watch the video
https://www.youtube.com/watch?v=xs9fddMg71o

Reference: Dweck, C., (2006). Mindset: The new psychology of success. Random House: New York.

Computer Logistics

Navigating the Internet

Students often struggle with typing in web addresses because of their limited typing skills and their lack of attention to detail. Many students would add spaces or extra punctuation to a web address, which resulted in an error message and much frustration and chaos in the classroom.



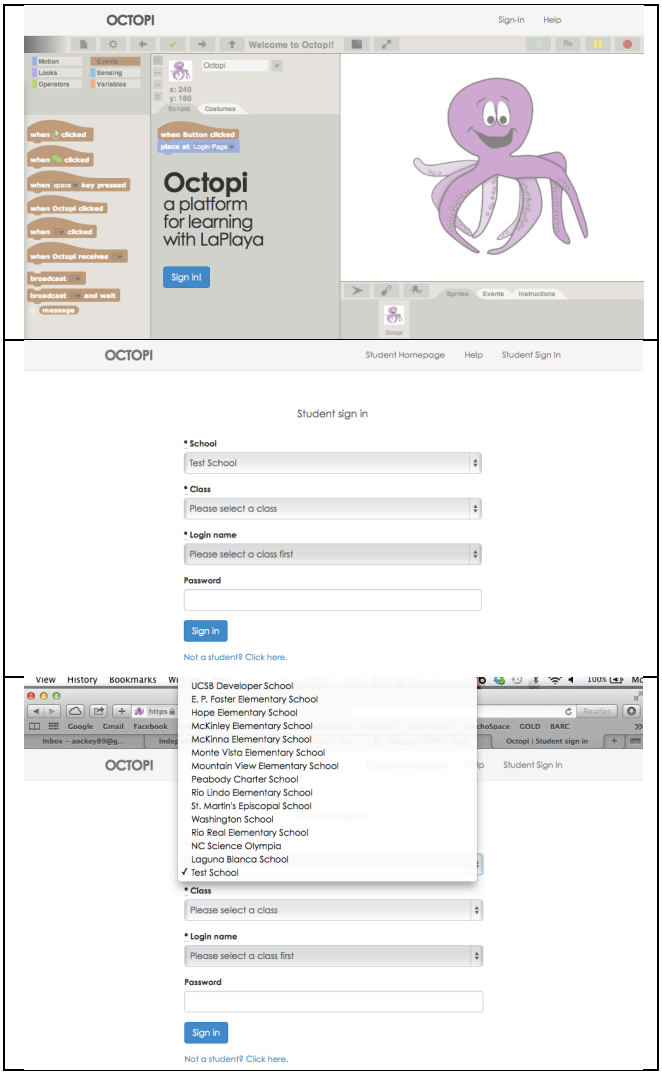
To help remedy this situation you can write the web address on the board in LARGE, clear print (You can even write it in all CAPS because you will navigate to the same page regardless of if the address is upper- or lower-case) and emphasize that there are NO spaces in web addresses.

This is also a good time to implement a system for what students should do if they have questions (if you have not already established this in your classroom) because this process usually results in multiple students having difficulty and since there is usually only one teacher students will have to wait before continuing on with the lesson.

Logging In to LaPlaya

Student Login-

Students should save their login username and password (you can have them write it inside their design notebook or somewhere else easily accessible). You may also want the students to bookmark the web address in their browser (Safari, Firefox, Google Chrome, etc.) so they can easily come back to the website without having to type the address in each time.



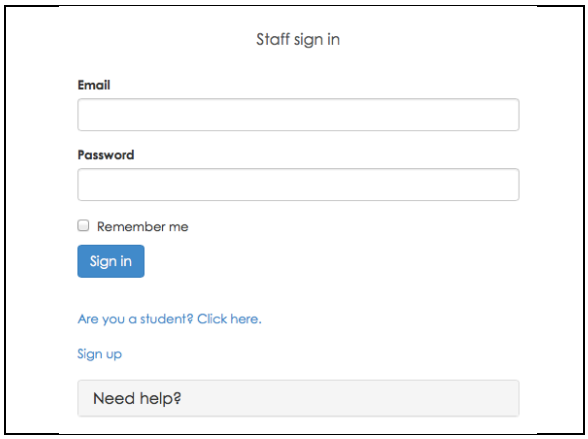
Once you get to the website you will see the LaPlaya interface in the background (top picture to the left). Students will click on the blue "Sign In" button to sign in to their account.

This will take you to a sign in page (middle picture to the left)where students will select their school from the dropbown menu (shown on the bottom picture to the left) and then select their class and login name (which will be assigned to them).

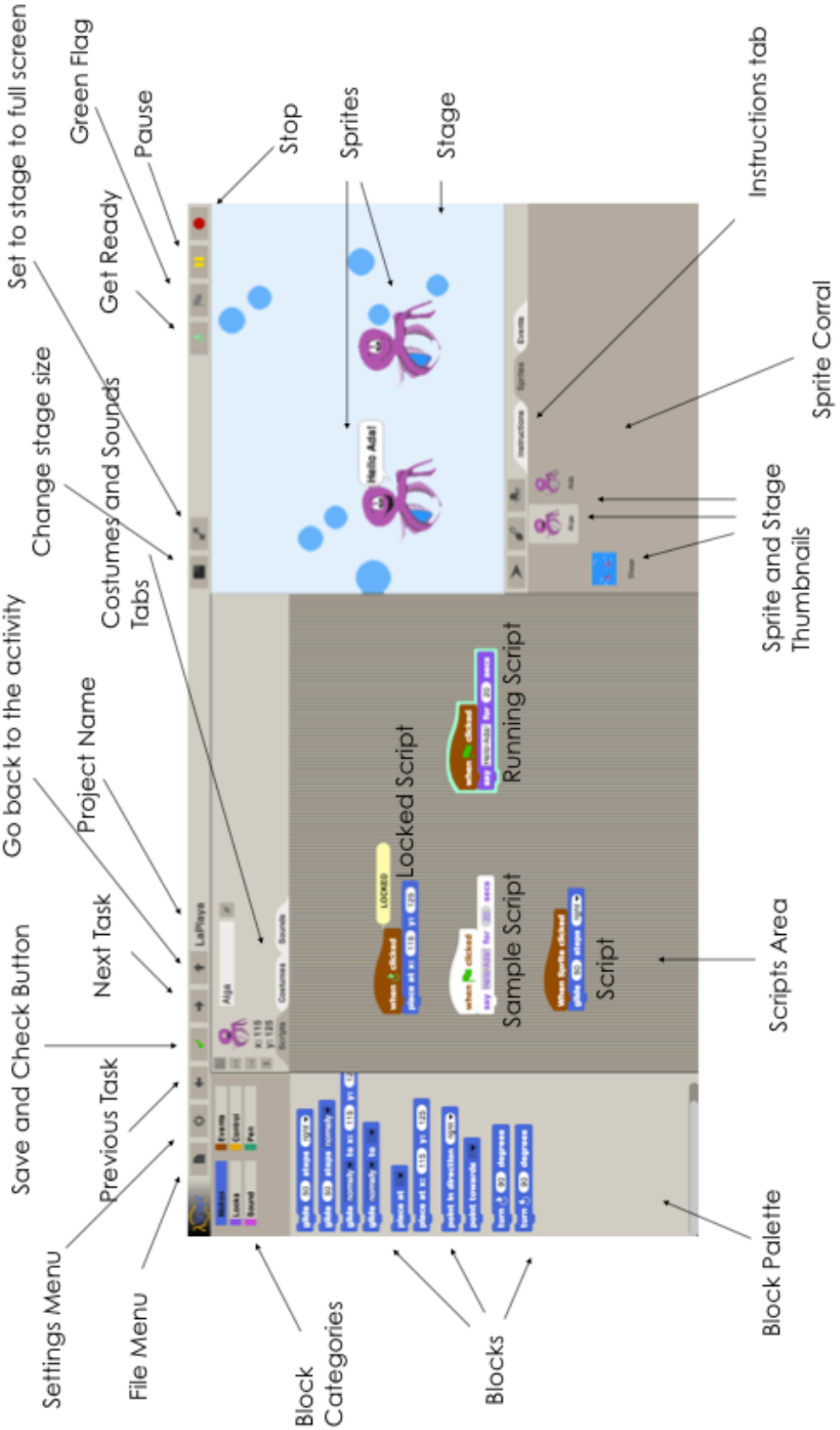
Students will also be given a password that they will need to type in. Their password should not be too difficult, but this is a good time to begin talking about how computer need you to give precise instructions and that it is important to type exactly what they are supposed to or the computer will not recognize it.

Teacher Login-

To sign in to a teacher account you will go to the same web address and click on the blue "sign in" button, but once you reach the log in page you will click on the blue writing at the bottom of the page that says, "Not a student? Click here." This will redirect you to the staff sign in page (shown to the right), where you will enter you email and password. Once signed in you have access to the curriculum, LaPlaya, and information about your class(es).



LaPlaya Interface Guide

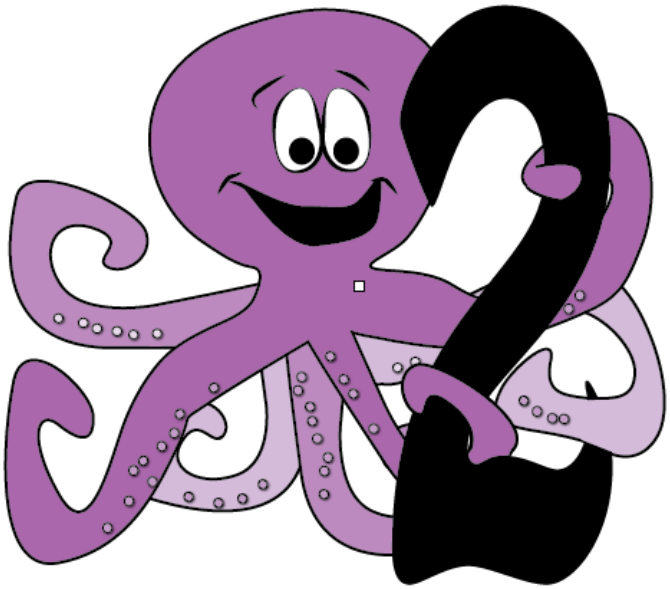


KELP-CS
UC Santa Barbara
2016

Module 2

Game Design

2015-2016



Diana Franklin
Computer Science Education
dmfranklin@uchicago.edu

Danielle Harlow
Science Education
dharlow@education.ucsb.edu

Table of Contents

Module 2: Game Design

Purpose

Module 2 is geared towards 5th grade students who have completed Module 1 (Digital Storytelling) and are already familiar with basic computational thinking and programming with our block-based, programming environment (LaPlaya). This module teaches students about loops, decision-making, message passing, variables, and the game design process. Further, students will learn more complex programming by implementing these ideas in LaPlaya. Through the module, students will learn about and develop skills using the engineering design process. As a culminating project, they will apply all they have learned to create a game.

Module 2 takes approximately 13 – 14 hours to complete. Some of the activities are conducted in the classroom while students complete the remaining activities on computers (Wired Up). There are also lessons throughout where students engage in the engineering design process in the classroom to create and complete their own game by the end of the module.

Overview of Lessons

Day	Activity	Location (Time)	Description
1	Introduction to Game Design	Classroom (45-60 min.)	This lesson introduces the main topic of Module 2: Game Design. It also introduces students to the idea of thinking about the user or the person/people they are designing for. In the first half of the lesson students are introduced to the idea of game design and what makes a good/fun game. They are also invited to think more like a game designer by brainstorming and discussing the kinds of things they would want to include in a game that they might invent. In the second half of the lesson students begin to differentiate between a game designer and a game user. Once a user has been identified (either by them or by the teacher) students will create a "User Bio" that includes basic information about their user that might affect the games they play. Lastly, students will come up with some ideas for games that their user

			would like to play, which will serve as a starting point for them to later brainstorm game ideas that they will design and program using LaPlaya.
2	Review of Module 1	Lab (45 min.)	This lesson give a brief review of the concepts covered in module 1.
3	Defining an Engineering Problem	Classroom (30-45 min.)	This lesson introduces students to the engineering problem that they will be solving through the programming of a game. The teacher gives students a topic that is phrased as en Engineering Problem (A science topic, vocabulary words, history content, etc.) and students use that to create their game. They also must think about the constraints (the LaPlaya programming environment, time, etc.) when programming their game. Students will then write a problem statement about what the problem is, and why it is important to solve.
4	Brainstorming Games	Classroom (45-60 min.)	In this activity students will be going through the brainstorming process to generate ideas for a game that they can program and then narrow them down to find the best solution. This process teaches students that all ideas are good ones, and those that may sound crazy might be the best ones. This also makes students evaluate their own ideas and come up with a rationale to pick the best one.
5	Broadcast & Receive	Lab (45 min.)	LaPlaya is an example of object-oriented programming, where objects are the focus of the program; they're the ones who are in control and make things happen. Object-oriented programming is great because it gives you an easy-to-understand strategy to for breaking up problems. For example, if you had a scene with a flying bird and a walking dog, the bird would be in charge of its flying and the dog would be in charge of its walking. They can

			work completely independent of each other, so if you decide that you'd rather have a bat instead of a bird, you can take out the bird and the dog would still be there, walking correctly. The tricky part of this scenario is how to get the objects to work together. This is a hard part of real life too, which you've probably seen for yourself if you've ever had to work on a group project before! When we work in groups, we communicate with each other so we know what the other person is doing and what our jobs are. In LaPlaya, the objects, sprites, communicate with each other by passing messages. A sprite can broadcast a message to the rest of the sprites, and other sprites can have scripts that run when they receive that language. The messages in LaPlaya aren't like the kinds of messages we use, like calling or emailing someone; they're only visible to the sprites, not to us. In the later tasks students will also learn how to create their own messages and what to do when more than one message is being broadcast simultaneously.
6	Game Design	Classroom (30-45 min.)	In this activity, students practice thinking about the game player (or user). In design, it is important to consider the interests, wants, and needs of who will be using your product. In this lesson, students will decide on the main aspects of their game. What is the main objective of the game? What are the rules of the game? What keys will the game player use to interact with the game? It is important to think through these design decisions ahead of time, and with the game player in mind.
7	Storyboarding	Classroom (30-45 min.)	This lesson gives students the opportunity to create a storyboard of the game they intend to create for this module. This gives students the

			opportunity to think about the big things they wish to include in their game and to think about what they want the game to look like to the user.
8	Flowchart Basics	Classroom (15-30 min.)	This lesson introduces flowcharts to students, which are used by computer programmers (among others) to help plan out their program and visually display all of the different parts of a program. This lesson bridges the conceptual divide between the storyboards students created previously and the sophisticated flowcharts that students will be creating in the next lesson that includes multiple sprites, variables, and loops.
9	Introduction to Loops	Lab (45 min.)	Repeat blocks are important because they let you be more concise: if you want to move 50 steps and then turn right 10 times, you could use a move 50 steps block followed by a turn right block, followed by a move 50 steps block, etc. ten times, but that's pretty tedious. In language we have shortcuts that tell you "Charlotte meant to write those blocks out 10 times, but that would be really annoying so instead she shortened it" and even though I didn't write it out, you still understand what I mean. We do the same thing in LaPlaya with repeat blocks. Repeat blocks are one kind of loop, and we'll see more later in module 2. "Loops" are called "loops" because, in general, they mean "do this thing once and then loop back to the top and do it again". In this case, you "do it again" the number of times listed in the block. Other loops have different ways of telling you when to stop, or not to stop at all!
10	Complex Flowcharts	Classroom (45-60 min.)	This lesson introduces more complex flowcharts that are used for programming. This lesson gives students a method for adding more complex concepts to their flowcharts, like loops

			and decisions, which they are learning in Module 2. The goal is to have students create more complex flowcharts of their game before they begin programming on the computer. This allows students to flesh out their game conceptually and to think about what kinds of features to include in their game.
11	Sensing & Decisions	Lab (45 min.)	The snake uses a different kind of loop, the forever loop block. This block tells the script to keep doing it forever. What would happen if you attached a block to the bottom of the forever loop block? You can't! LaPlaya doesn't let you attach one there because it knows the script will never get to it; it'll keep executing the loop forever so nothing can happen after that. In the wall we're using a forever block in a slightly less obvious way: instead of always doing something like moving, we're using it to always check for something. Once the green flag is clicked, the explorer will check over and over if it's touching the sprite "walls" and if it is, it'll say OUCH. The light blue touching block has a question mark in it because it's like LaPlaya is asking a question: is the explorer touching the walls? If it's true, then we do the block(s) inside of the if block. If not, then we don't.
12	Programming	Lab (TBD)	Students should be given some time on the computer to program the game that they have planned in the prior lessons. They can add to their programs as they learn more computer science concepts in the following wired-up lessons.
13	Introduction to Variables	Lab (45 min.)	In this activity, we'll be using variables to make a game. A variable is a place to store a value, either a word or a number. You can name a variable anything you want but it's helpful to give it a name to describe what it's storing. For example, if you want to use

			a variable to store how many points a player gets during a game, you could name it "Fred" but it would be more helpful to give it a name that tells you what it's storing, like "score". There's two types of variables: local (for this sprite only) and global (for all sprites). In this game, the variable is global so you can change it in any of the sprites. In the last task, we use an if/else block to determine what the background should look like. This is really similar to the if block we used earlier, but now we also get to specify what will happen if the if statement is not true — this is the part that goes in the else.
14	Full Project 1: Cat vs. Dog Game	Lab (45 min.)	*Initialization* Students should initialize all of the sprites; use the demo to decide which attributes need to be initialized for each sprite. *Cat throws fish* This is kind of tricky because what seems like one action — a cat throwing a fish — is actually broken up across two sprites. The cat needs to look like it's throwing the fish, but the fish actually "throws" itself; it needs a script that will make it move. It also needs to go back in the bag, so you need to move it back into the bag without the user seeing it happen on the screen. The other tricky part is that you're only using one fish but pretending like there's multiple fish in the bag. What happened if you used multiple fish sprites instead? How would the cat know which fish to throw? It's easier to only have one fish because then you're always throwing the same one, but then you can run into some problems, like what happens if you try to throw "another" fish before the fish hits the ground? *Dog movement* Use a forever block to make the dog walk back and forth forever. What should go inside the forever block? This

			can be tricky because it's not enough to just make it walk one way forever; it has to turn too. *One hit* Program the dog to flash if it gets hit. This is really similar to The Wall in sensing/decisions: you're using an if block inside of a forever to always check for a certain condition. *Three hits* Now you only want it to do something after three hits. How do you keep track of how many times it's been hit? You should use a variable, like in the Intro to Variables activity. In this project there's already a variable named points for you to use.
15	Test & Redesign	Lab (TBD)	Students will test their programs and identify potential issues or areas for improvement. Students will then redesign their programs and spend more time doing final programming of their games before sharing their work in the last lesson of the module.
16	Full Project 2: Falling Game	Lab (45 min.)	*Initialization* Initialize all of the sprites using the demo to figure out which attributes will change. The weird thing about this one is you have to initialize all of the falling objects to specific locations (which are in comments on each sprite). We want to start all of the objects at different heights so that when they fall, they won't all fall at the same time. *Falling objects* This is really similar to throwing the fish in the last activity: you only have one of each object, but you want to make it look like there's more than one. Program each to fall visibly and then move it back to its starting location without the user seeing it happen on the screen. *The crab* This is like the dog movement in the previous activity.

			Points This is like the one hit/three hits tasks in the previous activity.
17	Final Programming	Lab (TBD)	Students will be on their computers programming their game. Students should take the feedback they received from the previous lesson to help improve and reprogram their game before sharing it with the group.
18	Sharing	Lab (TBD)	In this lesson students will be sharing their final program (their game) with others.

1: Introduction to Game Design

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces the main topic of Module 2: Game Design. It also introduces students to the idea of thinking about the user or the person/people they are designing for. In the first half of the lesson students are introduced to the idea of game design and what makes a good/fun game. They are also invited to think more like a game designer by brainstorming and discussing the kinds of things they would want to include in a game that they might invent. In the second half of the lesson students begin to differentiate between a game designer and a game user. Once a user has been identified (either by them or by the teacher) students will create a "User Bio" that includes basic information about their user that might affect the games they play. Lastly, students will come up with some ideas for games that their user would like to play, which will serve as a starting point for them to later brainstorm game ideas that they will design and program using LaPlaya.

For an interesting take on the power of games on young people, and the future use of games to better society watch the TED talk from Jane McGonigal below:

http://www.ted.com/talks/jane_mcgonigal_gaming_can_make_a_better_world#t-1186617

Objectives-

Students will able to:

- Identify multiple games of different types and which aspects of those games are fun for users.
- Understand that all games were invented by someone for some purpose.
- Differentiate between thinking like a game designer and thinking like a game player (user).
- Identify a user (or users) that will be playing a game they design.
- Create a bio for their game user, which includes thinking about their likes, dislikes, and computer skills.
- Generate multiple ideas for games that their user would enjoy playing.

Materials-

Teacher	Student
	Thinking About the User Handout Design Notebook (optional)

Learning Tasks

(Total Approx. Time: 45-60 minutes)

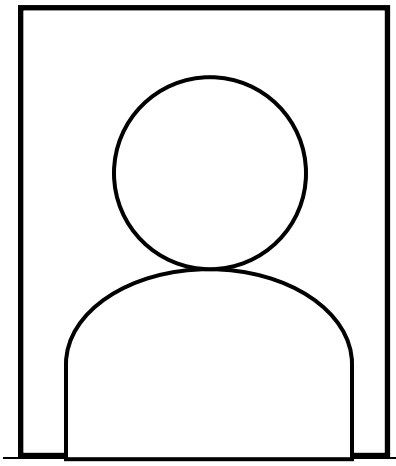
Approx. Time	**Learning Tasks**	**Purpose/Instructional Strategies**
15-20 min.	Discussion on Games & Game Design: • Discuss with the class what games are and have them brainstorm some examples of fun games. • Discuss what kinds of things make those games fun and why. • Inform students that someone invented all the games there are today. For example, Monopoly was originally invented in 1903 to educate and warn people about land monopolies before it became a hugely popular game (from the Wikipedia article on the game Monopoly). • Ask students to think about the kinds of things they might include if they were to invent a new game (you may wish for them to write these ideas down, talk with a partner, or call out to you).	*A discussion on what games are and what makes them fun will open up the module by getting students thinking critically and allowing them to get into the mindset of thinking like a game designer.* • *Games can include board games, video games, computer/phone games, playground games, etc.* • *It is important to be aware that girls may not participate as much because they tend to have less experience with computer/video games that may be brought up more frequently in discussion.* • *Thinking about what aspects of these games is fun will help students think about the kinds of things they should include in the game(s) they will be designing to create a better user experience* • *It is good to remind students that all the games they know were invented by someone for many different reasons and so they can create their own game as well.*
30-45 min.	Thinking About the User: • Discussion with students about what/who a user is (a user is a person or group of people that will	It is important for students to think about the user at the beginning of the design process as well as throughout. Empathizing with who will be using their program and how they will be using it is a large part of designing

	be utilizing what you are designing/making); in this case the user will be the game player. • Either you decide who the user will be or have students decide this. This user will be the person that students will be keeping in mind when designing and programming their game. This is an important part of the Engineering Design Process ("Thinking About the User") and will be referenced throughout the module. Students should fill out who their user is at the top of the student worksheet. • Work with students to help them fill out a user bio: Who their user is, their age or age range, their computer skills, their likes and dislikes. • In the "Game Ideas" section of the student worksheet, at the bottom of the user bio, students should generate a few initial game ideas that they think their user may enjoy playing. These ideas should not be well fleshed out yet, and may include games that already exist and they know the user enjoys (for example, if the user is their sister and they know she loves Candyland).	anything that others will be using. • It is good to discuss with students what a user is as a concept because it can be difficult for some students to think about designing for someone other than themselves. It is also important for them to think about the different kinds of users that are possible and to really think about how they will be using their program (what their limitations are, their likes and dislikes • It is important that a user (or users) are defined either by you or by the students so they have a concrete idea of who they are designing their game for throughout the module. • By having student create a bio for their user it has them think about who this person (or group) is and how THEY would use the computer and play their game. Without this step students may design a game that they would enjoy, but their user may not. • When students are thinking about the computer skills of their user it is important that they think about things like typing skills, whether there is a mouse or laptop track pad that is easy to click (some computers make it harder to click on things than others) or if the user will be playing alone or with someone else (in which case certain modifications may be considered). • Help students think about possible games that their user may enjoy. They may know their user personally and already know some games that they enjoy playing or they may have to user the likes and dislikes list that they create for the user bio to make some educated guesses about the kinds of things they might enjoy playing. This should give them a good idea of what kind of game they should design later on.

Thinking About the User

Creating a User Bio:

User: _____



Age: _____

Likes:

_____

_____

_____

Dislikes:

_____

_____

_____

Computer Skills:

_____

_____

_____

_____

Game Ideas:

_____

_____

_____

_____

_____

_____

_____

_____

Notes:

_____

_____

_____

_____

2: Review of Module 1

Teacher Lesson Plan

Lesson Rationale-

This lesson give a brief review of the concepts covered in module 1.

Objectives-

Students will be able to: • *Use multiple events from Module 1 in programs such as using when sprite clicked, keys, and when other sprite clicked.* • *Initialize multiple things in a program correctly.* • *Use costume changes to animate sprites.*

Materials-

Teacher (optional) Computer Projector (to demonstrate Game)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5-10 min.	*DEMO* Maze Game	This demo shows students the goal of the lesson and gives them an idea of what kinds of things they will be programming and learning how to do throughout.	**PLAY THE GAME!** Use the arrow keys to move the explorer through the maze. Click on the door to open it. Watch out for the snake.
	Task 1 Initialization	Students are learning to initialize a game so that each time a user wants to start playing their game it will be set up correctly each time.	Initialize the explorer's starting location and starting costume when the "Get Ready" button is clicked! ***HINT: The explorer should be at the same place as her shadow, and in "Costume 1".
	Task 2 Walking Explorer	Students are learning sequentially how to program each part of a complex game starting with programming the main sprite to respond to the user by using the "when key pressed"	Use the arrow keys to move the explorer! Program the explorer to: • ***Glide right when the right arrow key is pressed. • ***Glide left when the left arrow key is pressed. • ***Glide up when the up arrow

		block.	key is pressed. • ***Glide down when the down arrow key is pressed.
	Task 3 Animation!	Students are learning about animation by programming costume changes for sprites.	Make the explorer look like she's walking! Program the explorer to change costumes every time she moves.
	Task 4 *Open Door*	Students are learning how to enhance the user experience by incorporation more events when the user clicks on a sprite.	In order for the explorer to reach the treasure, the door needs to open! Program the door to open when clicked by using a "turn" block.
	Task 5 *Blocked Out*	Students are learning about how to make an action occur in one sprite when the user clicks a different sprite.	Make the rest of the maze visible when the door is opened! ***HINT: Switch the stage's background when the door is clicked.

Suggestions for Students who Finish Early-

- Play in the Sandbox (an open-ended version of LaPlaya that includes all of the blocks that students will be learning in this module).

3: Defining an Engineering Problem

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces students to the engineering problem that they will be solving through the programming of a game. The teacher gives students a topic that is phrased as en Engineering Problem (A science topic, vocabulary words, history content, etc.) and students use that to create their game. They also must think about the constraints (the LaPlaya programming environment, time, etc.) when programming their game. Students will then write a problem statement about what the problem is, and why it is important to solve.

Objectives-

Students will be able to:

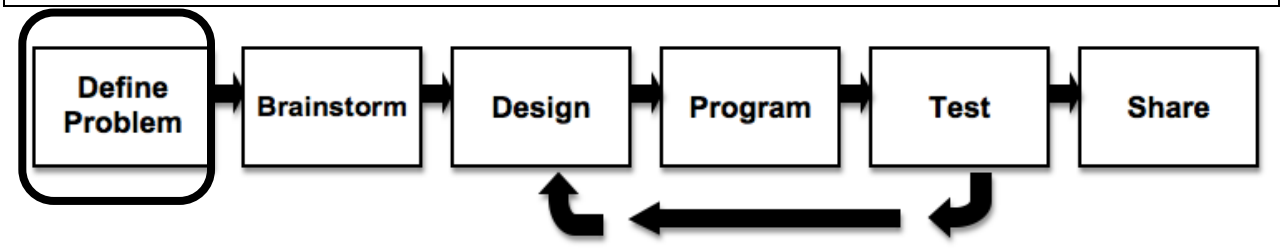
- Write a problem statement that includes who has a problem, what the problem is, and why it is important to solve.
- Identify constraints for their project.

Materials-

Teacher	Student Engineering Problem Worksheet Design Notebook (optional)

Thinking About the User-

When students identify the engineering problem they must think about who has the problem (the user) and what exactly their problem is. To do this it is important for them to remember their user bio from the previous lesson and identify the constraints for solving the problem using this information (likes/dislikes, computer skills, etc.).



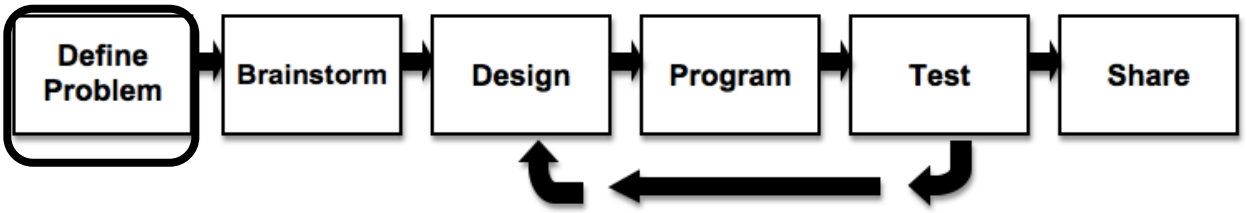
Learning Tasks -

(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Use the Engineering Design Thinking Process at the top of the student worksheet as a reference to introduce this first step in the Design Thinking process; Define a Problem. Connect this with the definitions at the top of their Engineering Problem worksheet.	By showing students where they are in the Engineering Design Thinking process throughout all the lessons it reinforces the parts of the design cycle and contextualizes them so students are always aware that they are working through the process themselves. Going over the vocabulary is also essential for helping students to understand the parts of an engineering problem and differentiating the constraints they must work within.
3-5 min.	Introduce the Engineering Problem to students. Some suggestions include: • A younger class needs to learn about natural disasters, but they get bored easily. How can we share what we know with them in a fun way? • I (the teacher) want to learn about the California Missions like I was actually there. How can we make them more interactive? Ideally, WHO has the problem will align with the user students created a bio for in the previous lesson. If not, you may consider having students create a new bio for this user.	By introducing an engineering problem in this way it gives students more of a purpose for creating their games. They also include all of the elements necessary for students to complete a problem statement: • Who has a problem • What is the problem • And, Why is the problem important You may have your students create a game for any sort of engineering problem (it is up to you), but the 3 parts of the problem statement (see above) should be clear to students. If the user is different from the one students created a bio for in the previous lesson you may want them to create a new bio for this new user, which will help them identify what kinds of things they will need to keep in mind while breaking down the engineering problem and identifying potential constraints.
10-15 min.	Have students complete #1 – 3 of the problem statement on their Engineering Problem worksheet.	This is the first part of defining an engineering problem, and students can and should refer back to this throughout the creation of their games.
10-15 min.	Help students identify the constraints they will need to work within when solving this engineering problem (They	The basic constraints include the LaPlaya programming environment and time (which is up to you), but students should also think about their user and constraints that may

	must use LaPlaya, they will have only a set amount of time, their user may have limited computer skills, etc.) Students can then complete #4 of the problem statement on their worksheet.	involve them (for example, their omputer skills, what they do and don't like, etc.) Let students be creative because they may come up with very interesting ideas. Helping students with this process is essential. These limitations should be explicit before they begin working on the more open-ended programming environment so they don't get lost in the many options available.
3-5 min.	Go over with students how they will be recording their data and progress throughout the design thinking cycle of programming their digital story (this is up to you).	t is crucial that students get into the habit of recording things and keeping track of what they have done, need to do, and will do. You may decide to have you students create a design notebook or simply a binder of the worksheet provided with some extra paper; How they record things is up to you.

Engineering Problem



What is an engineering **problem**?	Something that you, another person, or a group of people NEED. Sometimes you might identify a problem yourself or be asked by someone else to help solve a problem they have identified.
What are **constraints**?	LIMITATIONS that you must work within while trying to solve an engineering problem. This could be time, money, materials, or knowledge.

Problem Statement

1. WHO has the **problem** or need? *(Your teacher, you, someone else?)*

2. WHAT is the **problem** or need? *(Your teacher will tell you this)*

3. WHY is it important to solve? *(What is the **goal** of the project?)*

4. What **constraints** will I need to work within? *(Time, Knowledge of Programming, LaPlaya limitations, something else?)*

4: Brainstorming Games

Teacher Lesson Plan

Lesson Rationale-

In this activity students will be going through the brainstorming process to generate ideas for a game that they can program and then narrow them down to find the best solution. This process teaches students that all ideas are good ones, and those that may sound crazy might be the best ones. This also makes students evaluate their own ideas and come up with a rationale to pick the best one.

Objectives-

Students will be able to:

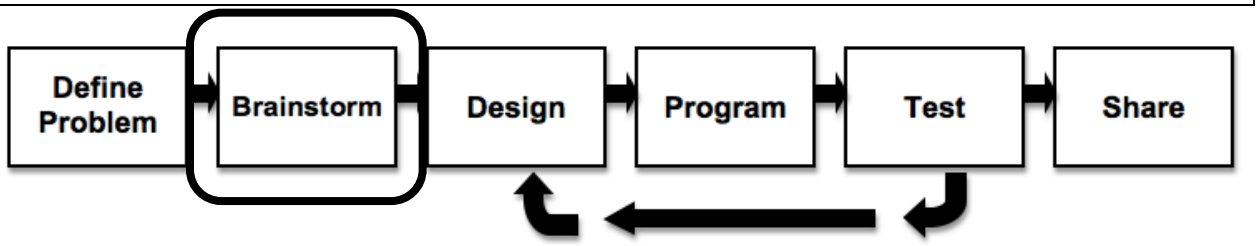
- Generate multiple ideas to solve the engineering problem presented to them.
- Use logical reasoning to narrow down their ideas.
- Work collaboratively to produce the best game idea possible.

Materials-

Teacher	Student
Timer (optional)	Brainstorming worksheet 8-10 small pieces of paper (per student) 5 half sheets of paper (per student) Design Notebook (optional)

Thinking About the User-

As students generate ideas for their games in this lesson they will need to keep in mind the user from the User Bio they create earlier. It is important that they come up with ideas for games that their user (or game player) would actually enjoy playing because ultimately the quality of a game is judged by the people who play it (users), no matter how much the game designer (them) may like it personally.



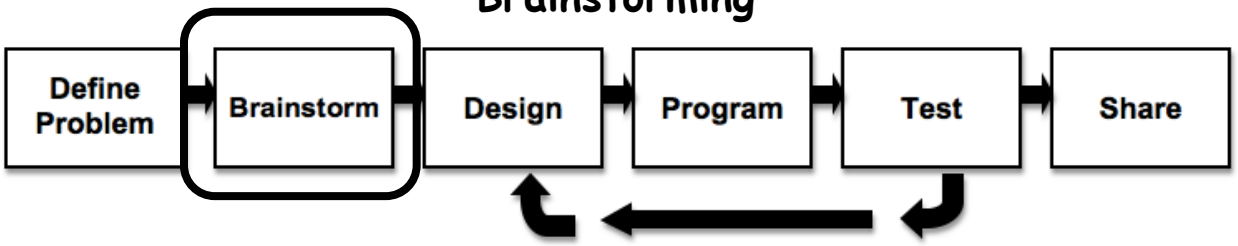
Learning Tasks -

(Total Approx. Time: 45-60 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Introduce the Brainstorming part of the Engineering Design Process. Refer to the diagram at the top of the student worksheet. Go over the definition of "Brainstorming" at the top of the worksheet with students and discuss with them that brainstorming is a multi-step process where they will be working independently and collaboratively.	Introducing the vocabulary and the contexts that brainstorming is used in will help solidify what the process is to students and going over the fact that the process of brainstorming involves working along as well as with others before students actually engage in it themselves should give them an idea of what is expected before beginning.
5-10 min.	Discuss guidelines for step 1 of the brainstorming process; Generate as many ideas as you can as quickly as you can. Emphasize that there are no bad ideas and that students can choose to write or draw ideas (or both) on their small sheets of paper, but they don't need to be pretty or descriptive at this point. Also, remind students to keep their user in mind when coming up with ideas because their games will only be good if their user enjoys playing them. You may want to time students and challenge them to come up with the most ideas at the end of that time. We suggest about 5 minutes for this 1st round of idea generation.	In this step it is important that students understand that their ideas, writing, and drawings of those ideas do not need to be too descriptive or perfect. It is important that students come up with many ideas for their games so they have a larger pool to chose from when picking their final, best idea to create their program on. Students can be messy when writing and drawing their ideas on their paper, what it important is that they know what the ideas are so they can evaluate them in the next step of the process. It's good for students to be reminded of their user throughout the Engineering Design Thinking process, especially at this stage because ultimately they are coming up with ideas for a game that someone else (their user) will play and the game is only as good as the user thinks it is (harsh, but true).
3-5 min.	Have students evaluate their ideas and write 3 to 5 of their best ones on the given lines in step 2 of the worksheet. To evaluate their ideas have students think about the	By doing this students will need to take into consideration the engineering problem and constraints as well as their user when picking their best ideas and come up with logical reasoning as to why some ideas may be better than others. They may want to refer

	Engineering Problem they are solving, the constraints they are working within, and their user.	back to the Engineering Problem worksheet as well as the User Bio they created.
5-10 min.	Model as a whole class how to do step 3 of the brainstorming process; Building on ideas. Pick an idea that was generated (use this as an example) and make that idea better and more elaborate by getting suggestions from students and thinking collaboratively.	Building and elaborating on their ideas may be difficult for students so going through this process as a whole class may help them understand how to do this. This exercise will also give them practice at working in a collaborative environment and providing constructive feedback to their peers to help better their ideas, which they will be doing in step 3 of the brainstorming process.
5-10 min.	Pass out the half sheets of paper for students to write/draw their ideas on for this step. Have students work in pairs or small groups to build and elaborate on their top 3-5 ideas (from step 2). You may want to walk around and help students with this process if it appears that they are having difficulty.	This process helps students build on ideas and think about how they can make ideas better. It also gives them practice working collaboratively and giving and receiving feedback. This skill is important and can be applied to other people's ideas as well allowing students to think critically about everything around them. You may want to discuss that this is what is done all the time; people build on other's ideas and make them better (ex. The iPhone used others' ideas about touch screens and combining internet/phone capabilities and put them together in a new, better way).
5-10 min.	Have students go through the new, improved ideas they generated in step 3 and evaluate them to pick the single best idea. You may want students to keep working in pairs or small groups to determine which of their ideas is the best. Make sure to have students think about their user when picking this final idea because this will be the game that they are programming for that user. Students will then write this final game idea on step 4 of their worksheet.	Once again, students are taking into consideration the engineering problem and constraints as well as their user when evaluating their own ideas and narrowing them down to the best single idea they generated. This process also gives students valuable practice working collaboratively (an important part of science and engineering) as well as providing constructive feedback to others and receiving feedback positively. Giving/receiving feedback may be something you choose to explicitly discuss with you class as this may be difficult for students to do effectively. The final idea students pick will be the game that they create using LaPlaya during this module.

Brainstorming



What is **brainstorming?**	A technique to generate IDEAS and come up with CREATIVE solutions to problems. It started in advertising and is now used for business, research, writing, design, and much more.

Step 1: Think about the **Engineering Problem** you are solving and the **User** that you are designing this game for. Come up with as many ideas for your game as you can! All ideas are good ideas!

DRAW or WRITE your ideas on the small pieces of paper your teacher gives you. One idea per piece of paper.

Step 2: Pick 3-5 of your ideas that you think are the best. WRITE these below.

1. _____
2. _____
3. _____
4. _____
5. _____

Step 3: Use the ideas you picked above and make them BETTER and more detailed. Work with your classmates and THINK about the questions below.

- *How can you make your ideas better?*
- *Can you combine any of your ideas?*
- *Can you add to an idea to make it more interesting?*

DRAW or WRITE your ideas on the pieces of paper your teacher gives you. One idea per piece of paper.

Step 4: Use the work you just did to pick your BEST idea for a game. This is the game that you will be programming in LaPlaya. WRITE the idea below.

_____

_____

5: Broadcast & Receive

Teacher Lesson Plan

Lesson Rationale-

LaPlaya is an example of object-oriented programming, where objects are the focus of the program; they're the ones who are in control and make things happen. Object-oriented programming is great because it gives you an easy-to-understand strategy to for breaking up problems. For example, if you had a scene with a flying bird and a walking dog, the bird would be in charge of its flying and the dog would be in charge of its walking. They can work completely independent of each other, so if you decide that you'd rather have a bat instead of a bird, you can take out the bird and the dog would still be there, walking correctly.

The tricky part of this scenario is how to get the objects to work together. This is a hard part of real life too, which you've probably seen for yourself if you've ever had to work on a group project before! When we work in groups, we communicate with each other so we know what the other person is doing and what our jobs are. In LaPlaya, the objects, sprites, communicate with each other by passing messages. A sprite can broadcast a message to the rest of the sprites, and other sprites can have scripts that run when they receive that language. The messages in LaPlaya aren't like the kinds of messages we use, like calling or emailing someone; they're only visible to the sprites, not to us. In the later tasks students will also learn how to create their own messages and what to do when more than one message is being broadcast simultaneously.

Objectives-

Students will be able to:

- *Use broadcast and receive to send invisible messages between multiple sprites.*

Materials-

Teacher (optional) Computer Projector (to demonstrate)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
	DEMO *Traffic Lights*		Click on the traffic light to make the car drive home!
	Task 1	Students are introduced to the idea of broadcasting and	**Messages are necessary when an action to one**

	Receiving	receiving messages between sprites. Students start by programming a sprite to receive a message that is already being broadcast by a different sprite.	**sprite (light) causes an action in another sprite (car)** ***Program the ***car*** to drive home when the ***traffic light*** is clicked! ***This requires two steps: 1) When light is clicked, it broadcasts a message 2) When car receives the message, it drives home. ***We have already programmed the light. You need to program the car. ***HINT: Look at the white example script in the car's scripts area.
	Task 2 Sending	Students are learning how to coordinate messages between multiple sprites to enable an action in one sprite to occur when something happens to a different sprite.	Try programming the whole thing this time! ***Program it so when someone clicks the traffic light, the car drives to the target. Do both parts now: • ***When the stoplight is clicked, it broadcasts the "Green Light" message. • ***When the car receives that message, it drives to the target. ***HINT: Make sure the car stays on the road!
	Task 3 Parallel	Students are learning how to broadcast and receive complex messages between multiple sprites that must occur in a particular sequence.	Oh, no! There are ***two traffic lights*** and ***two cars!*** ***When the ***left*** traffic light is clicked, the ***blue*** car drives in front of the ***blue*** house. ***When the ***right*** traffic light is clicked, the ***yellow*** car drives in front of the ***yellow*** house. ***Hint: You need two

			different messages, one for each traffic light! ***Hint: Make sure your messages have different names!
	Task 4 Crossroads	Students are learning how to program complicated broadcast and receive messages that may occur simultaneously in or a particular order between multiple sprites.	Make both cars go to their targets when the traffic light is clicked! *** Remember what you've learned! You can do it!
	Task 5: BONUS Traffic Jam!		Make both cars go to their targets when the traffic light is clicked!

Suggestions for Students who Finish Early-

- Play in the Sandbox!

6: Game Design

Teacher Lesson Plan

Lesson Rationale-

In this activity, students practice thinking about the game player (or user). In design, it is important to consider the interests, wants, and needs of who will be using your product. In this lesson, students will decide on the main aspects of their game. What is the main objective of the game? What are the rules of the game? What keys will the game player use to interact with the game? It is important to think through these design decisions ahead of time, and with the game player in mind.

Objectives-

Students will be able to:

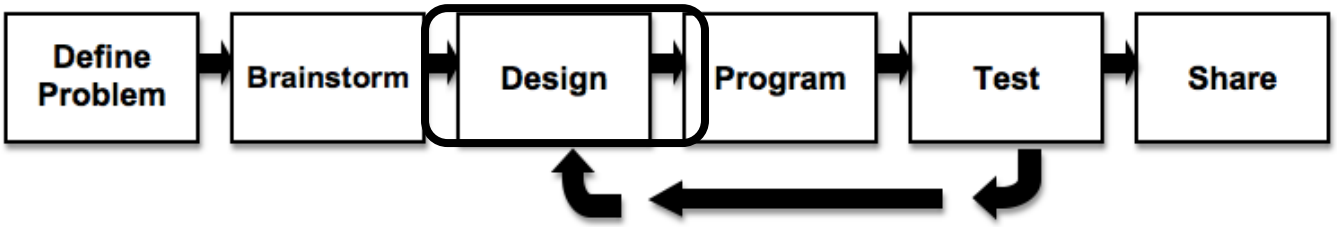
- Create objective and rules to play their game.
- Determine which event blocks they should use so the user can interact with their game.

Materials-

Teacher	Student Design Notebook (optional)

Thinking About the User-

Students will need to think about the user by determining the objective and rules for their game that the user must follow. They will also determine how the user will interact with the game by deciding which event blocks will trigger actions. By stating these things explicitly it will be easier for students to find a way to communicate these things to the user during game play.

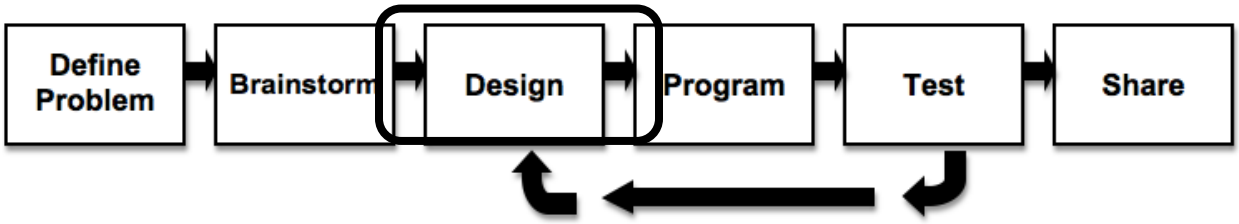


Learning Tasks -

(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
10 min.	Select a simple game that most students will know the instructions to play (e.g. Steal the Bacon, Heads Up-Seven Up). Have students discuss how to play the game, as a group or with a partner. Then, work together as a class to write detailed and succinct instructions to play the game for somebody who has never played.	This serves as a warm-up, and an opportunity for students to practice writing detailed game instructions with guidance from the teacher. Young students often struggle to produce clear, succinct and sequential instructions. Model how to do this, and your thinking behind each decision.
5 min.	Transition students to thinking about their own games, and their previously selected game player. Have students create instructions to play their game (the OBJECTIVE).	
5 min.	Have students think about the rules of their game and write 3-5 in the lines provided on their "Game Design" worksheet.	
5 min.	Lastly, have students consider how the game player will interact with the game. Which keys will trigger which actions? They will circle their choices in step 3 of the "Game Design" worksheet.	Students should write or draw what each even block will do in the blank space provided to the write of each option. For the "when sprite clicked" and "when key pressed" options they should write which sprite or key they are referring to on the lines provided.
2-3 min.	After students have completed the "game design" worksheet have them brainstorm what other kinds of things they should include in their game that they may not have included in the worksheet.	For example, students may wish to include different event blocks in their game than the ones provided (like when other sprite clicked or broadcast/receive). They may also have more rules for their game than they provided in the worksheet, etc.

Game Design



What is Game Design?	The process of planning the CONTENT and RULES for the game. A game designer should consider: • The OBJECTIVE for the game player. (Example: Get the explorer to the treasure without touching the snake!) • The RULES for playing the game. (Example: Don't hit the walls of the maze.) • HOW the user will interact with the game. (Example: Use the arrow keys to move the explorer.)

Step 1: Think about the **Engineering Problem** you are solving and the **User** that you are designing this game for. What is the **Objective** for your game? DRAW or WRITE your ideas here!

Step 2: What are the rules of your game? Write 3-5!

1. _____

_____

2. _____

_____

3. _____

_____

4. _____

_____

5. _____

_____

Step 3: How will the game player interact with the game?

Circle the EVENT BLOCKS you will use below. Write or draw what will happen for each one you plan on using in the blank space provided (pick at least 4)

Blue Square (Get Ready)	Green Flag
When Sprite Clicked Which sprite? _____ _____	When Key Pressed Which key? _____ _____

7: Storyboarding

Teacher Lesson Plan

Lesson Rationale-

In this activity, students practice thinking about the game player (or user). In design, it is important to consider the interests, wants, and needs of who will be using your product. In this lesson, students create a storyboard *from the perspective of the user*. What will the game player see when the game starts, or ends? How will the game player learn the instructions to play the game? It is important to think through these design decisions ahead of time, and with the game player in mind.

Objectives-

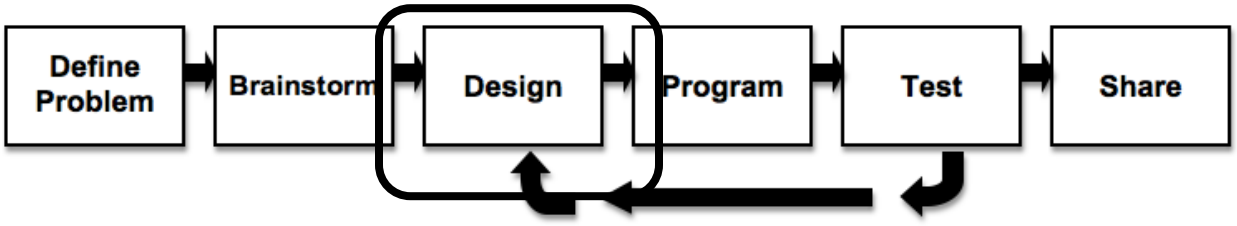
Students will be able to: • Create a storyboard for the game, from the perspective of the user.

Materials-

Teacher	Student Design Notebook (optional)

Thinking About the User-

This lesson has students think about the user by having them visualize what their game will look like to the user when it is being played.



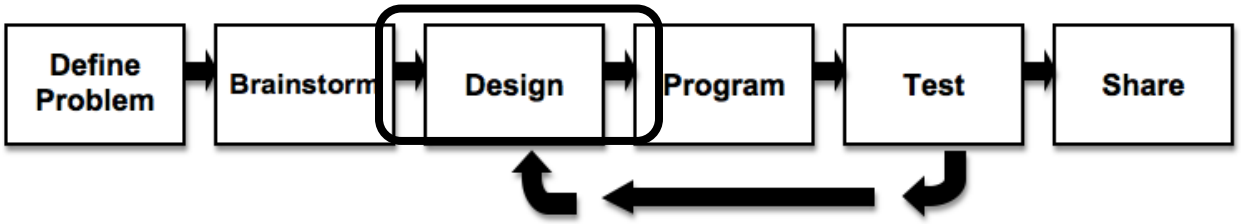
Learning Tasks -

(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Briefly go over what storyboards are and how they are used in comic books, movies, etc. (optional to show a storyboard to students that	It is important to go over the terminology with students and providing an example gives them a familiar context to connect what they are learning about to. If you show an example of a storyboard you may want

	they may be familiar with (find some examples online).	to find one for a children's movie that students will most likely have seen before (Toy story, Frozen, etc.)
3-5 min.	Discuss with students what the main parts of their storyboard will include (Introduction to the game, teaching the player how to play, main events, ending screen)	It is important for student to think about all of these aspects of their game and how they can make their game user-friendly by providing information on how to play the game, what will happen under certain circumstances (like if the player gets a penalty), and what will happen at the end if the player wins or loses the game.
30-45 min.	Have students create the flowchart of their game. Students should only spend about 5-10 minutes per section or frame and should be encouraged to not spend too much time on any single aspect of their storyboard.	It is important for students to really flesh out their game from the user's perspective and think about how they will communicate things to the user the most effectively, however students should not be worried too much about what their drawing look like or spending too much time on these storyboards because they will simply be used as a guide for their game.

Storyboarding



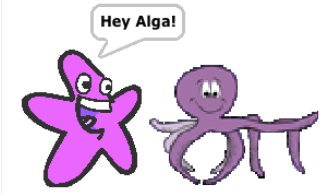
What are **storyboards?**	A series of sketches with written descriptions that represent the important parts of a planned program. They are often used for movies, television, and even comics.	
What are the parts of a **storyboard?**	**Introduction:**	Set the scene for your story. Introduce your characters and establish the setting. This should also include instructions on how to play your game that the user can easily understand and follow.
	Memorable Moments:	Something important that happens to your character (they go somewhere, do something, or meet someone). *You can have more than one memorable moment.*
	Conclusion:	A final screen that communicates to the player that the game is over. This may include a message telling them that they won or lost the game or a message for users.

Remember, a storyboard has 2 main parts:

1. **The SKETCH of the main events in the story.**
2. **The written DESCRIPTION of what is happening.**

See the example here → → →

Introduction



Alga the Octopus and Marty the Starfish are friends with a nice life in the ocean.

KELP-CS
UC Santa Barbara
2014

Name/Number: _____ Page # ____ of ____

Storyboard for the story about _____.

Write the part of the story (Introduction, Memorable Moment #__, Conclusion) that each scene is describing on the line above it-

_____	_____	_____
Description: _____	Description: _____	Description: _____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

8: Flowchart Basics

Teacher Lesson Plan

Lesson Rationale-

This lesson introduces flowcharts to students, which are used by computer programmers (among others) to help plan out their program and visually display all of the different parts of a program. This lesson bridges the conceptual divide between the storyboards students created previously and the sophisticated flowcharts that students will be creating in the next lesson that includes multiple sprites, variables, and loops.

Objectives-

Students will be able to:

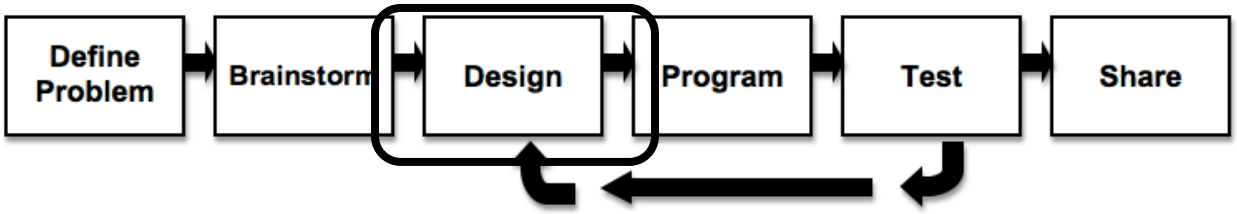
- Identity the different parts of a flowchart and understand what each means.
- Create and interpret a basic flowchart.
- Interpret a program's flowchart for what the user will experience.

Materials-

Teacher	Student
Whiteboard (or other way to draw simple example flowchart with class)	Flowcharts Worksheet Design Notebook (optional)

Thinking About the User-

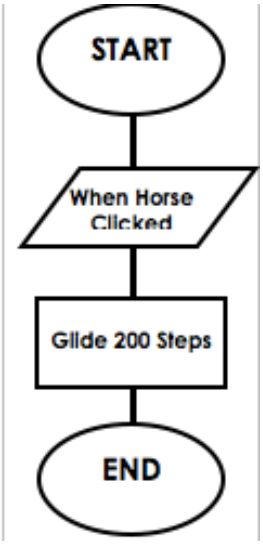
After a simple example flowchart is created as a class the students will be asked to interpret this flowchart from the user's perspective. They will be asked what the user will see when the program runs, what actions the user does that trigger events, as well as making a judgment about whether the user will enjoy this program.



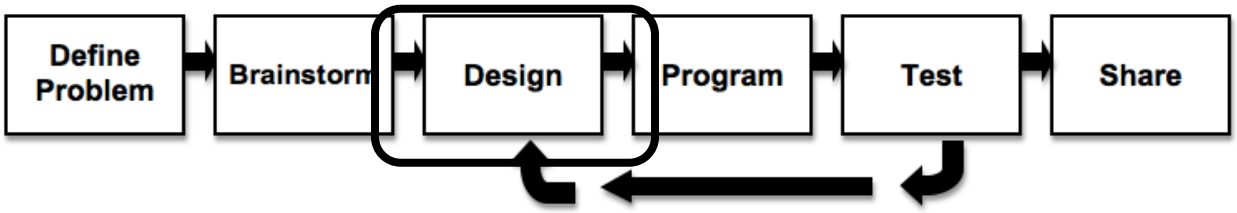
Learning Tasks -

(Total Approx. Time: 15-30 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
3-5 min.	Go over what flowcharts are and how they are used in programming. Emphasize that all software developers	Going over what flowcharts are and how they are used in computer science is important for

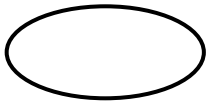
	(programmers) go through this process to layout what the parts of their program are and how everything works together before even getting on the computer.	students to understand the importance of this process in the engineering design cycle before getting on a computer to program their games.
15-20 min.	Go through what the different parts of a flowchart are (lines, dotted lines, rectangles, ovals, and parallelograms) and go through an example flowchart that either you come up with or have students think of a basic scenario that they can create a flowchart of (for example, when you click on the horse sprite it runs a face—see flowchart below). Have students identify what each part does and what the shapes mean. 	It is important to spend some time going though the different parts of a flowchart with students because they can be confusing and might not make sense out of context (when students are not on the computer). By having students go through an simple example flowchart, like the example provided here, they can begin practicing making flowcharts, determining which shapes must be used for which parts of the program, and figuring out the flow of the final program.
3-5 min.	Go through the flowchart you created with the class and discuss how the user will experience this program. What will the user do that will start the program, what will the user see when the program runs, will the user enjoy this program, how could it be better?	It is important for students to not only think about their flowcharts like a programmer, but also think about how their planned program translates to the user. It is also good for them to think about whether the user will enjoy the program.

Flowcharts



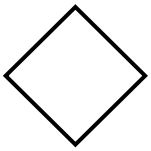
What are **flowcharts?** A way to organize and show the "flow" of your program. Programmers use flowcharts to work out the steps in a program they are designing before programming anything on the computer.

What are the parts of a **flowchart?**


The START or END of a program.


A PROCESS that will be performed. (Ex. "Glide 50 Steps")


An INPUT that starts a process. (Ex. "When Sprite Clicked")


A DECISION that is built into the program (Yes or No)

The RELATIONSHIP between the parts of a program.

The RELATIONSHIP between different sprites (Ex. "When Sun Clicked" - - "Boy says It's hot!")

9: Introduction to Loops

Teacher Lesson Plan

Lesson Rationale-

Repeat blocks are important because they let you be more concise: if you want to move 50 steps and then turn right 10 times, you could use a move 50 steps block followed by a turn right block, followed by a move 50 steps block, etc. ten times, but that's pretty tedious. In language we have shortcuts that tell you "Charlotte meant to write those blocks out 10 times, but that would be really annoying so instead she shortened it" and even though I didn't write it out, you still understand what I mean. We do the same thing in LaPlaya with repeat blocks. Repeat blocks are one kind of loop, and we'll see more later in module 2. "Loops" are called "loops" because, in general, they mean "do this thing once and then loop back to the top and do it again". In this case, you "do it again" the number of times listed in the block. Other loops have different ways of telling you when to stop, or not to stop at all!

Objectives-

Students will be able to:

- Use loops to repeat actions of sprites
- Use loops in conjunction with motion in sprites
- Make a distinction between a forever loop and a repeat loop

Materials-

Teacher (optional) Computer Projector (to demonstrate Task 1)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*Task 1* Triangle		***Loops help you do the same thing with fewer lines of code. They ***repeat*** the same thing several times. ***The white example script draws a triangle without using loops, but the script is very long. ***Shorten the script by using a **LOOP** ("***repeat*** __ ***times***" block) to draw a triangle when the pen is

			clicked.
	Task 2 Square		Draw a square when the pen is clicked! Remember to use a loop. ***Hint: The angle of a square is 90 degrees. Don't draw off the screen!
	Task 3 Hexagon		Draw a hexagon! Remember, a hexagon has 6 sides. ***HINT: The angle of a hexagon is 60 degrees. Don't draw off the stage!
	Task 4 Party		Use loops to make the ballerina dance when she is clicked. Look at the Cool Dude's script for an example. ***HINT: Remember to use wait blocks in between costume changes!
	Task 5 Bonus Star		This is a difficult project! Draw a star when the pen is clicked. Remember to use loops. HINT: Try different degrees to make the star!

Suggestions for Students who Finish Early-

- Play in the Sandbox!

10: Complex Flowcharts

Teacher Lesson Plan

Lesson Rationale-

Students will be learning about more complex flowcharts that include more complicated programming concepts, such as loops and decisions. Students will then get practice adding these kinds of complex aspects to a flowchart of a common board game and then begin creating a flowchart for the game that they are designing and programming themselves in this module.

Objectives-

Students will be able to:

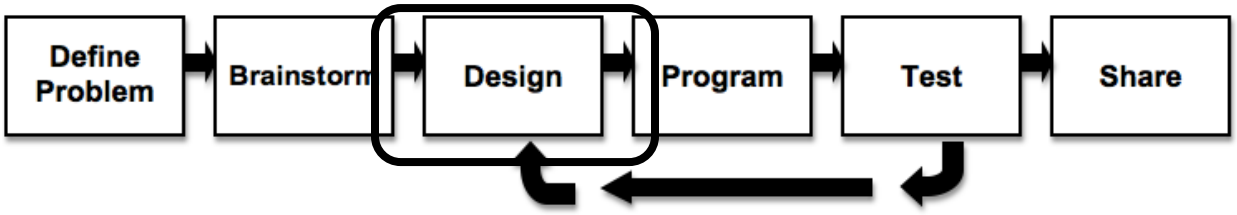
-

Materials-

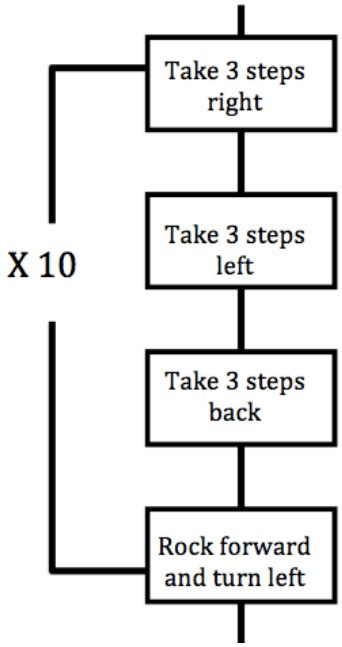
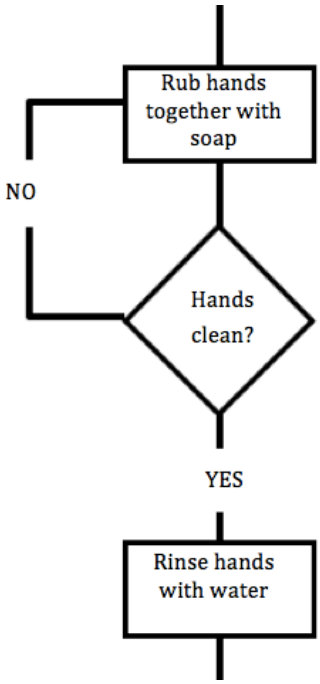
Teacher	Student
Projector/white board to show example of complex flowcharts (see loop and decision examples below).	Large piece of paper/poster paper Pen/Pencil Design Notebook (optional)

Thinking About the User-

By completing more complex flowcharts of their games, students will be thinking about how to program the features of their game that the user will see and use (such as loops and decisions). By completing these flowcharts students will get to visualize all of the aspects of their game and figure out how each part of their program will work together.

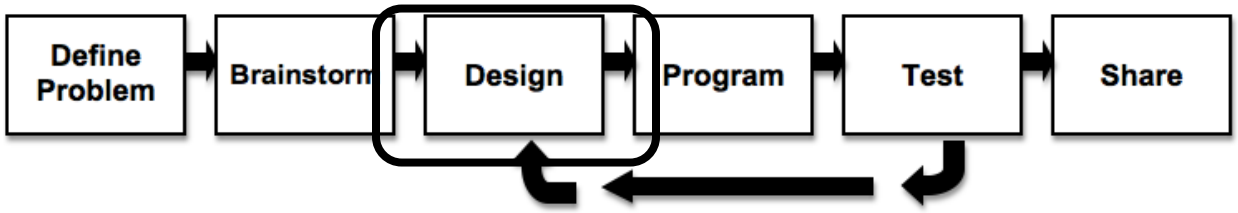


Learning Tasks -
(Total Approx. Time: 45-60 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
5-10 min.	Discuss the new, more complex things that students can use when programming (broadcast & receive, loops and decisions specifically) and how students can add these things to flowcharts. Have them refer back to their "flowcharts" worksheet that goes over the different parts of a flowchart and how each one related to different aspects of LaPlaya: • Dotted lines between sprites should represent interactions between sprites that "when other sprite clicked" and "broadcast/receive" • Loops can be shown by having a solid line loop around a series of blocks or a script with a box in the middle saying how many times to repeat the loop (see example at right). • Decisions can be shown with a diamond with the question in it and two lines coming off of it showing the next step depending on if the answer was "yes" or "no" (see example at right). Have students use the blank flowcharts provided in the "complex flowcharts" worksheet to draw an example of these three types of complex flowcharts	Flowchart for the "grapevine" line dance, a loop (below)  Flowchart for washing your hands, a decision (below). 

15-20 min.	Have students decide on (or pick yourself) a simple game that the whole class is familiar with. Students will work together to create a flowchart for this game that should include some decisions and loops and if there are multiple characters or players will involve showing the interactions between them. Students will work together to create a simple flowchart for their game on a large piece of paper/poster paper.	An example of a simple game that students can create a flowchart for includes a sport like baseball, soccer, etc. or a board game like Candyland or shoots and ladders. These games shouldn't be too complicated and students should not spend too much time creating a flowchart for them; this is just to get them familiar with how to add complex aspects to flowcharts. Student shouldn't cerate this flowchart on their own because it will be too complex, but rather you should encourage them to work together in pairs or small groups to figure out all the little pieces they should include in the flowchart.
30 min.	Once students complete the flowchart for the simple game have them start drawing a flowchart for the game that they are designing and programming in this module; They may want to start with the first scene from their storyboard and determine what they need to include to make those things happen and translate this into a flowchart.	By breaking up their flowchart into different section it may be less overwhelming for students to think about and draw out their program. Emphasize the importance of creating flowcharts as an essential skill that all game designers and programmers do in the field.
	If time, or in future days have students complete their flowchart for their entire game. They should be able to come back to their flowchart, add things, change things, etc. throughout the module when they are working on their game.	This may be something that you want students to work on in their design notebook so it is easier for them to go back and refer to the flowchart and add to/modify it easily.

Complex Flowcharts



Create a flowchart for a game that has 3 sprites, characters, or players.

Show how these sprites/characters/players interact & include at least 1 decision that they must make in the game.

Broadcast & Receive	Loops	Decisions
START	START	START
END	END	END

11: Sensing & Decisions

Teacher Lesson Plan

Lesson Rationale-

The snake
The snake uses a different kind of loop, the forever loop block. This block tells the script to keep doing it forever. What would happen if you attached a block to the bottom of the forever loop block? You can't! LaPlaya doesn't let you attach one there because it knows the script will never get to it; it'll keep executing the loop forever so nothing can happen after that.
The Wall
Now we're using a forever block in a slightly less obvious way: instead of always doing something like moving, we're using it to always check for something. Once the green flag is clicked, the explorer will check over and over if it's touching the sprite "walls" and if it is, it'll say OUCH. The light blue touching block has a question mark in it because it's like LaPlaya is asking a question: is the explorer touching the walls? If it's true, then we do the block(s) inside of the if block. If not, then we don't.

Objectives-

Students will be able to:

- Use a forever loop in a program and understand that this script does not have an end.
- Program aspects of a program to use sensing to trigger an action.

Materials-

Teacher (optional) Computer Projector (to demonstrate Game)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*DEMO* Maze Game		**PLAY THE GAME!** Get the explorer to the treasure. Use the arrow keys to move. Don't forget to click on the door to open it. Watch out for the snake!
	Task 1 Snake		Now, we're going to **PROGRAM THE GAME** for somebody else (a user) to

			play! Let's start by programming the snake. ***Copy the snake's white example script. The snake should move back and forth across the path. ***HINT: Use a "forever" loop to make sure the snake keeps moving!
	Task 2 Wall		You can also use forever loops to *sense when something happens*! *** Make the explorer say "Ouch!" when she runs into the wall! ***Hint: Look at the white example script. You need three parts for this: ***1) ***Forever loop*** that constantly checks ***2) Something you're ***waiting for*** (running into the sprite "wall") ***3) Something to ***do*** when it happens
	Task 3 Restart		Now, let's make a penalty for the game player (or user). Program the explorer to go back to the START of the maze if she hits a wall. ***Hint: Remember the three parts of a *sensing loop:* ***1) ***Forever loop*** that constantly checks ***2) Something you're ***waiting for*** (running into the sprite "wall") ***3) Something to ***do*** when it happens
	Task 4 Error!		Make sure the game player (or user) knows when the explorer hits a wall.

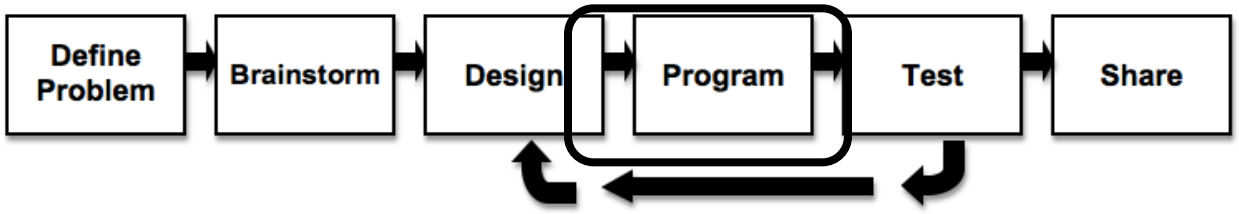
			Make the maze flash RED when the explorer hits a wall. ***Hint: Remember the three steps to a *sensing loop* ***Hint: Look at the *backgrounds* of the maze stage to make it flash red ***Hint: You *sense* in the explorer, but the *action* (background change) happens in the stage. You need a ***message!***
	Task 5 Victory!		When the explorer reaches the treasure chest, the game player (or user) wins! Let the user know the game is over by programming the explorer to *say "Victory!"* when she *reaches the treasure chest*. ***Hint: Remember the three steps to the *sensing loop!*

Suggestions for Students who Finish Early-

• Play in the Sandbox!

12: Programming

Teacher Lesson Plan



Lesson Rationale-

Students will be on the computer programming their game during time the teacher gives them for a reasonable amount of time to present a game ready for testing in future lessons.

Objectives-

Students will be able to: • Program their own game using LaPlaya

Materials-

Teacher	Student Design Notebook (optional) Computer

Thinking About the User-

Students must be thinking about their user throughout the programming of their game.

Learning Tasks -

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
TBD by teacher	Provide students with time on the computer to create a first draft of their game. In a future lesson these games will be tested and then redesigned and programmed into a final version suitable for sharing with others.	

13: Introduction to Variables

Teacher Lesson Plan

Lesson Rationale-

In this activity, we'll be using variables to make a game. A variable is a place to store a value, either a word or a number. You can name a variable anything you want but it's helpful to give it a name to describe what it's storing. For example, if you want to use a variable to store how many points a player gets during a game, you could name it "Fred" but it would be more helpful to give it a name that tells you what it's storing, like "score". There's two types of variables: local (for this sprite only) and global (for all sprites). In this game, the variable is global so you can change it in any of the sprites.
Lost or won?

In this task, we use an if/else block to determine what the background should look like. This is really similar to the if block we used earlier, but now we also get to specify what will happen if the if statement is not true — this is the part that goes in the else.

Objectives-

Students will be able to:

- Use variables to keep score in a game

Materials-

Teacher (optional) Computer Projector (to demonstrate Game)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*DEMO* Exercise Game		**PLAY THE GAME!** Press the "Green Flag" button to initialize the program and begin.
	Task 1 *New Variables*		If a program needs to remember anything - numbers, names, etc. - it needs ***variables***. This program needs to remember the score, so we created a variable named score. ***Part of the game has been programmed. Let's ***initialize*** the score variable by adding two blocks to the Get Ready blue-square script.

			• ***Add a block that ***sets score to 0*** • ***Add a block that ***to show the variable score*** (this shows it on the screen) ***Hint: All scripts may be programmed under the "PLAY" sprite.
	Task 2 Food and Drink		You made a variable ***score*** that starts at 0. Now add scripts to add or substract to the score as people play. • ***Program the game to INCREASE the score by 1 point when the player clicks on a HEALTHY food sprite! • ***Program the game to DECREASE the score by 1 point when the player clicks on an UNHEALTHY food sprite! ***Hint: A white example script is given for water. You can copy that to start. ***Hint: All scripts may be programmed under the "PLAY" sprite.
	Task 3 Activities		Program the game to keep score for activities! • ***Program the game to INCREASE the score by 2 points when the player clicks on a HEALTHY activity sprite. ***Program the game to DECREASE the score by 2 points when the player clicks on an UNHEALTHY activity sprite.
	Task 4 Lost or Won?		Let the game player (or user) know if they won or lost! We have two backgrounds - a won or lost background. ***If the score is **less than 5** after the message 'End' has been

			received, show a "lost" background. Otherwise, show a "won" background. ***Hint: Background stages occur in the stage. The script is started for you. Just fill in the rest! ***Hint: Look at the Operators and Variables categories

Suggestions for Students who Finish Early-

- *Play in the Sandbox!*

14: Full Project 1

Teacher Lesson Plan

Lesson Rationale-

Initialization
Students should initialize all of the sprites; use the demo to decide which attributes need to be initialized for each sprite.
Cat throws fish
This is kind of tricky because what seems like one action — a cat throwing a fish — is actually broken up across two sprites. The cat needs to look like it's throwing the fish, but the fish actually "throws" itself; it needs a script that will make it move. It also needs to go back in the bag, so you need to move it back into the bag without the user seeing it happen on the screen. The other tricky part is that you're only using one fish but pretending like there's multiple fish in the bag. What happened if you used multiple fish sprites instead? How would the cat know which fish to throw? It's easier to only have one fish because then you're always throwing the same one, but then you can run into some problems, like what happens if you try to throw "another" fish before the fish hits the ground?
Dog movement
Use a forever block to make the dog walk back and forth forever. What should go inside the forever block? This can be tricky because it's not enough to just make it walk one way forever; it has to turn too.
One hit
Program the dog to flash if it gets hit. This is really similar to The Wall in sensing/decisions: you're using an if block inside of a forever to always check for a certain condition.
Three hits

Now you only want it to do something after three hits. How do you keep track of how many times it's been hit? You should use a variable, like in the Intro to Variables activity. In this project there's already a variable named points for you to use.

Objectives-

Students will be able to:

- Program an entire game from start to finish using the concepts they have learned throughout Module 2

Materials-

Teacher (optional) Computer Projector (to demonstrate Game)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*DEMO*		**PLAY THE GAME!** Click the "Get Ready" and

	Cat vs. Dog Game		"Go" buttons to begin. The cat drops a fish from the bag when the space bar is clicked. Hit the dog with 3 fish to win the game!
	Task 1 Initialization		Initialize the starting score to 0 and starting location and costume for the dog, the cat, and the fish! HINT: The game player (or user) shouldn't be able to see the fish when the game starts.
	Task 2 *Instructions*	This is teaching a practice of providing instructions for users so they know how to play your game. There are several ways you can do this, but we will be using an initial instruction screen with a "play" button.	The player needs to know how to play the game! 1) Make sure the instructions screen shows first, with the play button. 2) When the player presses the play button, move into the game.
	Task 3 *Starting Game*		Now, you need to program the "Play" button! When the player clicks the Play button, * background changes to the tree * cat appears * dog appears * "Play" button disappears
	Task 4 Cat Throws Fish		Make the cat look like he's dropping the fish when the player presses the space bar! Then, make the fish appear and drop to the grass. HINT: Don't forget to put the fish back "in the bag" after it hits the ground so the cat can throw it again!

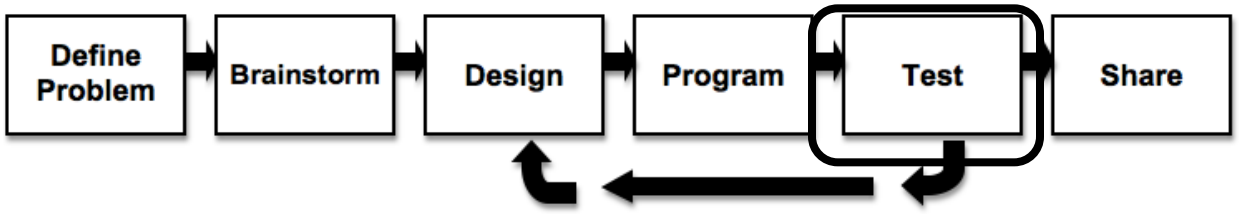
	Task 5 Dog Movement		Make the dog look like he's walking! HINT: He should start walking when the green flag is clicked. HINT: Walking requires **movement** and **costumes**.
	Task 6 One Hit		Program the dog to **flash** if he gets touched by a falling fish! HINT: Look at the **sensing** category. HINT: "Flashing" uses hide and show.
	Task 7 Three Hits		1.) Program the dog to count how many times he gets hit by a fish! Add 1 point each time the fish touches the dog. 2.) Program the dog to disappear after 3 hits! 3.) Program all scripts to stop after 3 hits! HINT!: To stop all the scripts use the stop block in the event block scripts!

Suggestions for Students who Finish Early-

- *Play in the Sandbox!*

15: Test & Redesign

Teacher Lesson Plan



Lesson Rationale-

Students will be working with each other to test their games (they will be utilizing their classmates to be "test users") and to give feedback for improving their game (either by making the programming better or by making it more user-friendly).

Objectives-

Students will be able to: • Give and receive constructive feedback about their work • Utilize feedback given by game testers (fellow classmates) to improve and redesign their games

Materials-

<u>Teacher</u> Stopwatch/timer	<u>Student</u> Computer Post-it notes Pen/pencil Design Notebook (optional)

Thinking About the User-

Students will be having users (other students) play their game to test how well it works and get feedback on what could be improved. Students will also be redesigning and reprogramming their games based on the test users experiences.

Learning Tasks-

(Total Approx. Time: 30-45 minutes)

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
2-3 min.	Have student log in to LaPlaya and open their game in the full screen mode (so a user can play it). Give each student a stack of post-it notes for their computer and instruct the class that each student needs a	Make sure that students know ahead of time that they will need to have their game ready for "test players" before this class period. Students should also be aware that their finished game should be able to be played in full screen mode without the user seeing any of the scripts or programming,

	pencil.	which is why they are testing games in full screen mode.
20-30 min.	Have students rotate around the room moving from computer to computer in an orderly fashion playing their classmates' games. Students should only be given a few minutes to play each game before they must provide feedback and move to the next computer. You can use a stopwatch or look at the clock to give then deadlines on when they need to move on (ex. 3 minutes to play the game, 2 minutes to give feedback, then move to the next computer)	Make sure that all students in the class are aware of what good feedback is. You may wish to have them use the sandwich technique where you say something that can be improved sandwiched between two things that you like about it. This should ensure that feedback isn't too negative and that students do not get hurt or offended by any of the feedback they receive on their game.
	Once a student finishes playing a game they will write their feedback (anonymously) on one of the post-it notes provided at that computer and stick it to the edge of the computer monitor (but making sure the screen isn't obstructed) Once they have been given a minute or so to provide feedback have the whole class rotate and begin playing the next student's game.	Depending on the time you have in the computer lab students may be able to play all of their classmates' games or they may only get to play a few, but the feedback that will be provided to each student will be valuable in helping them redesign and make their game better.
3-5 min.	When time is up in class and everyone has played at least a few of their classmates' games have students return to their original computer and collect the post-it notes that are stuck to the monitor. Remind students that these are opinions from "test players" of their game and they can be used to improve their game in the next class session.	Students should save these post-it notes either in their design notebook or on a separate sheet of paper. You may also want students to take notes on the feedback they were provided on their game by having them write the feedback in a column to the left of their paper and write how they can address this feedback in a column to the right of their paper (students can fold their paper in half to make the columns more defined).

16: Full Project 2

Teacher Lesson Plan

Lesson Rationale-

Initialization
Initialize all of the sprites using the demo to figure out which attributes will change. The weird thing about this one is you have to initialize all of the falling objects to specific locations (which are in comments on each sprite). We want to start all of the objects at different heights so that when they fall, they won't all fall at the same time.
Falling objects
This is really similar to throwing the fish in the last activity: you only have one of each object, but you want to make it look like there's more than one. Program each to fall visibly and then move it back to its starting location without the user seeing it happen on the screen.
The crab
This is like the dog movement in the previous activity.
Points

This is like the one hit/three hits tasks in the previous activity.

Objectives-

Students will be able to:

- Program an entire game from start to finish using the concepts they have learned throughout Module 2

Materials-

Teacher (optional) Computer Projector (to demonstrate Game)	Student Computer

*Learning Tasks (*Total Time: 45 minutes)

Approx. Time	Learning Tasks	Purpose	Student Directions
5 min.	*DEMO* Falling Game		**PLAY THE GAME!** Make sure you have the Falling Game Observation Sheet to fill out!
	Task 1 Initialization		First, make sure everything is ***initialized*** correctly when the "Get Ready" blue square is clicked! ***Sprites: Initialize location and hide/show ***Variables: Initialize value and hide/show

			***Stage: Initialize background ***Hint: Make sure you have your Falling Game Observation Sheet!
	Task 2 *Start Game*	This is teaching a practice of providing instructions for users so they know how to play your game. There are several ways you can do this, but we will be using an initial instruction screen with a "play" button.	Now let's change scenes to get into the game! When the play button is pressed, start the game! ***Hint: This is easier if you filled out your Falling Game Observation Sheet!
	Task 3 Falling Objects		Program the urchins and obstacles to fall! • ***Make the urchins, anchor, and wheel look like they're falling down the stage when the green flag is clicked. • ***Make the urchins, anchor, and wheel return to their starting location after falling down the stage (and hide!). ***Hint: Use the different speeds to make them fall differently ***Hint: Use wait blocks to offset when they are falling ***Hint: What do you use to make it happen over and over and over again?
	Task 4 The Crab		Program the crab! • ***Make the crab move left when the left arrow key is pressed ***Make the crab move right when the right arrow key is pressed.
	Task 5		Now, program what happens when the crab hits things.

	Collisions		Make sure you have your Falling Game Observation Sheet. ***What happens when the crab touches an urchin? (Urchin, crab, points) ***What happens when the crab touches the anchor or wheel (sprite, crab, points) ***Hint: Remember the 3 parts of the *sensing loop*! ***Hint: Don't forget to update the points!

Suggestions for Students who Finish Early-

- *Play in the Sandbox!*

Falling Game Observation Sheet

Part 1: Initialization

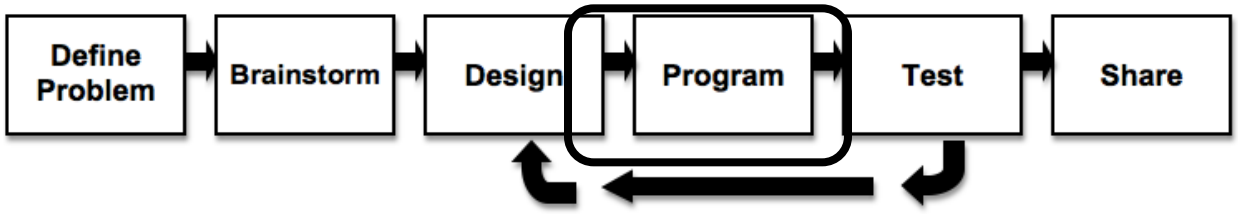
1) Draw the **background** you see first:	2) When you first click the **blue square**, which **sprites** do you see? *CIRCLE them below.* crab green urchin wheel anchor purple urchin play button 3) What do you **click** on to **start** the game? _____ 4) What **variable** can you see? _____ What **number** is it at the start? _____

Part 2: Game Play

1) Which sprites are **hidden** when you start the game? *CIRCLE them below.* crab green urchin wheel anchor purple urchin play button	2) For each **sprite** that is falling, CIRCLE the speed it is falling. green urchin Fast Medium Slow purple urchin Fast Medium Slow wheel Fast Medium Slow anchor Fast Medium Slow
3) What happens when the crab touches an urchin? The urchin _____ _____ The crab _____ _____ The points _____ _____	4) What happens when the crab touches the anchor or wheel? The anchor or wheel _____ _____ The crab _____ _____ The points _____ _____

17: Final Programming

Teacher Lesson Plan



Lesson Rationale-

Students will be on their computers programming their game. Students should take the feedback they received from the previous lesson to help improve and reprogram their game before sharing it with the group.

Objectives-

Students will be able to:

- Take feedback from others to program an improved version of their game
- Create a final version of their game that a user will be able to play

Materials-

Teacher	Student Design Notebook (optional) Computer

Thinking About the User-

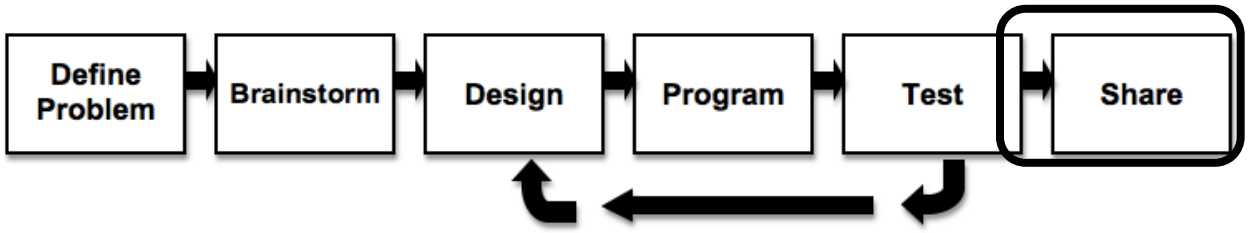
Students will need to take the feedback provided by test users (their classmates) in the previous lesson to help improve their games.

Learning Tasks-

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
TBD by teacher	Allow students some time on the computer to finish programming their games before sharing the final product in the final lesson of the Module. Ideally, students will take into account the feedback they received from their classmates in the previous lesson to improve and redesign their games.	

18: Sharing

Teacher Lesson Plan



Lesson Rationale-

Students will be doing a "gallery walk" of each other's games. This is also an opportunity to invite other classes or parents to see the students' games that they created and have the opportunity for people to play the games as well.

Objectives-

Students will be able to: • Share their finished games with others

Materials-

<u>Teacher</u> **Determine who/where your students will be sharing their games with.	<u>Student</u> Computer

Thinking About the User-

Students will be having other people (users) play their games.

Learning Tasks-

Approx. Time	Learning Tasks	Purpose/Instructional Strategies
TBD by teacher	In this lesson it is up to you to decide whom you wish to share your students' games with. You may just want them to save their game to the LaPlaya website (in which case they would be sharing their games with us, the curriculum developers), other students in the class (you could do a "gallery walk" where students can share and play their games with each other), another class, parents, or some	This lesson is dependent on the teacher to decide WHO the finished games students create will be shared with. It is important to communicate with students the importance of sharing their work with others. Emphasize that in engineering you are making a product for a user (in this case their game) and the end goal is to have a finished product that the user can use… other examples of this include engineering designing technology such as cell phones, computers, buildings, etc. that eventually have to be completed and used by others.

	other group.	