National Aeronautics and Space Administration

NEXT GEN STEM

COMMERCIAL CREW

Crew Orbital Docking Simulation (CODing Sim)

www.nasa.gov

For more about Next Gen STEM visit http://www.nasa.gov/stem/ccp.

Crew Orbital Docking Simulation

Overview: In this activity, students use Scratch, Snap!, or another programming language to create an interactive simulation of a spacecraft docking to the International Space Station. The Crew Orbital Docking Simulation (CODing Sim) engages students in computational thinking, problem-solving, and real-world applications of mathematics.

Grade Level: **5th-12th**

Suggested Time: **1hr - 2hrs**

Materials:

- ☐ *Earth Stage – download jpg (2.3 MB)*
- ☐ *Boeing CST-100 Starliner Sprite – download png (1.1 MB)*
- ☐ *SpaceX Crew Dragon Sprite – download png (2.3 MB)*
- ☐ *Space Station IDA Sprite – download png (1.2 MB)*
- ☐ *Computer(s) or tablet(s) with internet access (or download all content to run locally)*

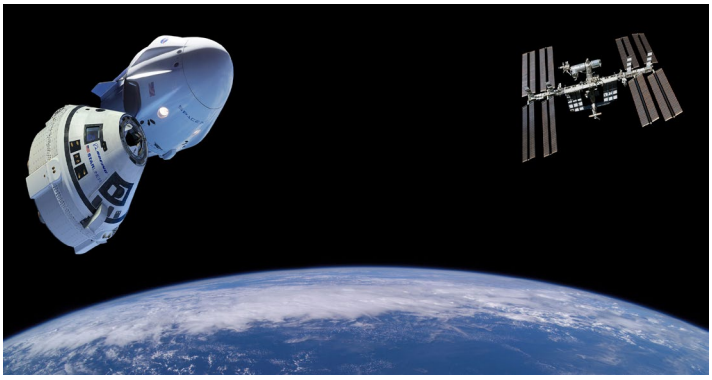
Programming language of your choice:

- ☐ *Free Scratch account at http://scratch.mit.edu*
- ☐ *Free Snap! account at https://snap.berkeley.edu/*
- ☐ *Scratch Advanced CODing Sim Example – download sb3 (2.2 MB)*
- ☐ *Snap! Advanced CODing Sim Example – download xml (1.2 MB)*
- ☐ *Scratch Beginner CODing Sim Example – download sb3 (0.9 MB)*
- ☐ *Snap! Beginner CODing Sim Example – download xml (1 MB)*

Common Core Standards for Mathematics (CCSS):
Practice: MP1, MP2
Content: 5.G.A.2, 6.RP.A.3, 7.EE.B.4, 8.F.B.4
High School: Modeling

ISTE Standards for Students (ISTE):
3a – Knowledge Constructor
4d – Innovative Designer
5d – Computational Thinker

What is a stage or sprite?



Management

1. This activity guide includes recommendations for **beginner** and **advanced** level programming. The advanced option requires more complex and sophisticated code, encourages higher-level thinking, and is more appropriate for students with experience using block-based programming languages. The teacher may also choose to select a different combination of program requirements based on the skill level of the students and the focus of the lesson.

2. The CODing Sim may also serve as a **culminating activity** after students have spent time learning about the various command blocks. Check out the **Additional Resources** for other activities to practice block-based programming. Then introduce the CODing Sim to the class as an opportunity to apply what they have already learned and demonstrate their understanding of the programming language.

3. It is important to emphasize that the simulation will only do what is programmed in the scripts. Remind students to use a **hat block** which is found in the Control/Events folder at the top of every stack of blocks. As students create code, allow them to **explore** what happens when they try different command blocks. Students might even simplify code by creating new command blocks. Encourage creativity.

4. It is good practice for students to take turns **beta testing** other students' programs to check if they are working as planned. Beta testing should take place throughout the program development. Student beta testers can provide feedback about any bugs or errors found while running the program including details of what actually happened versus what they expected to happen. Beta testing can be done by clicking the green flag while running the program in editor mode or in full screen mode.

5. Allow time for students to **reflect** on what they have learned and any challenges they encountered. Use class discussion or journaling throughout the activity to document problems, brainstorm solutions, and assess student progress.

6. Assessment:

- Does the program meet all of the requirements?
- Does the program do what it is supposed to do (simulates a spacecraft docking with the space station)?
- Have all errors or bugs been eliminated through code?
- Did the programmer(s) go beyond the basics and explore the addition of creative or more in-depth scripts?

7. There are hardware applications that are compatible with Scratch and Snap!. These extensions can add another element to the CODing simulation. Students might build a physical model controlled by the code or use physical sensors that trigger an action within the code.

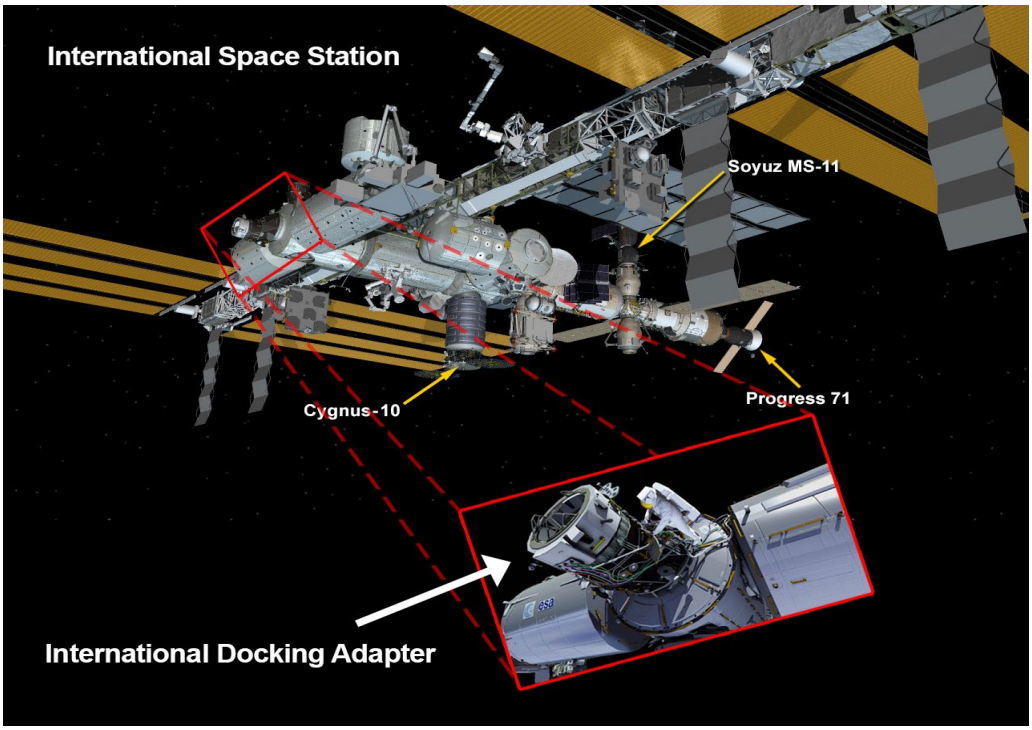
8. This activity guide is divided into multiple sections to provide **background** and **tips** for coding. It is not meant as a step-by-step guide or tutorial as there are many ways to reach the same outcome. Every student or group may create different code to meet the requirements. Use as little or as much of the guide as necessary, depending on student and teacher familiarity with block-based programming.

Jump to:

Section 1

Background Information and Engaging the Learner

The International Space Station is an orbiting laboratory located about 400 km above Earth where NASA learns about exploration as astronauts live and work in space. The International Docking Adapter (IDA) is a physical connecting point for visiting spacecraft and serves as both a parking spot for the vehicle and a gateway into the space station. In August 2016, the first IDA was installed on Node 2, the Harmony module, in preparation for Boeing's CST-100 Starliner and SpaceX's Crew Dragon, the Commercial Crew Program (CCP) spacecraft. To learn more about NASA's Commercial Crew Program, check out the CCP Primer at www.nasa.gov/stem/ccp.



Section 2

Getting Acquainted with Block-based Programming

The following websites provide guides, examples, and tutorials to help you and your students become familiar with the programming language.

https://scratch.mit.edu/ideas
https://snap.berkeley.edu/index.html#examples
https://snap.berkeley.edu/snapsource/help/SnapManual.pdf

About the Stage

The stage is the background layer of the project. Students may use the provided Earth backdrop or choose an appropriate image of their own.

Scratch Stage: 480 × 360 pixels
Snap! Stage: 480 × 360 pixels (default size)
Earth Backdrop.jpg: (dimensions: 1920 × 1440 pixels)

About the Sprites

A sprite is an object which performs functions controlled by scripts. The Commercial Crew spacecraft are sprites because they will need to move across the stage to dock with the space station. The IDA is also a sprite, so the size can be adjusted and the sprite can be moved to the front layer allowing the simulation to look more realistic.

Boeing CST-100 Starliner Sprite

Space Station IDA Sprite

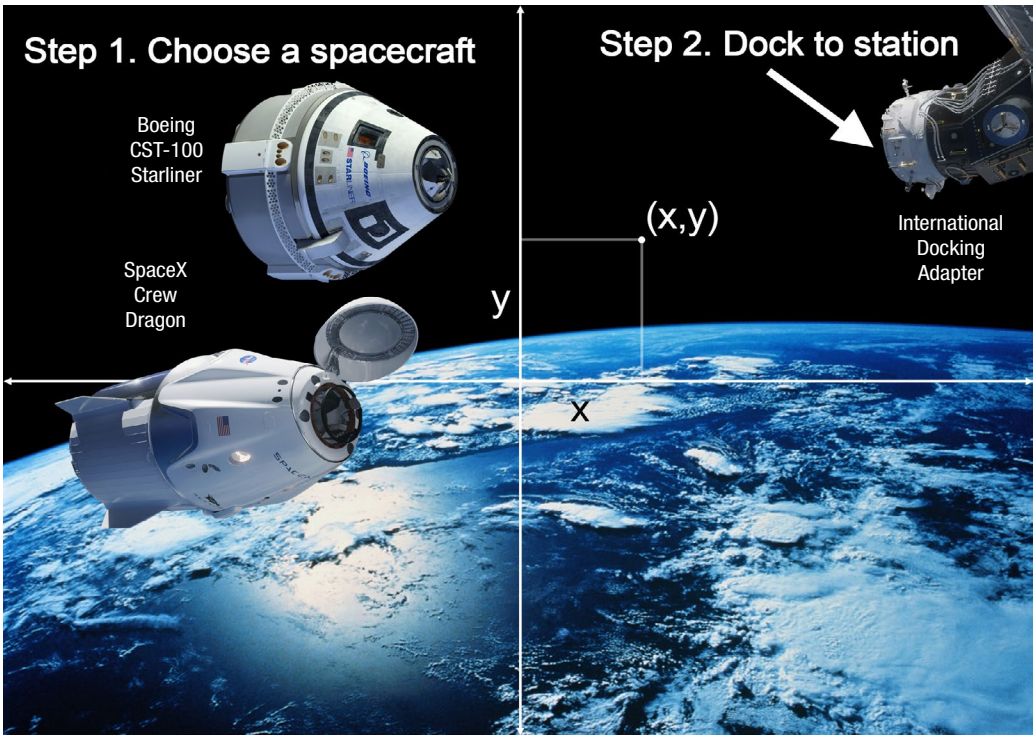
SpaceX Crew Dragon Sprite

Example of a hat block which begins a stack of blocks or script. The hat blocks can be found in the Control or Events folder.

About the Scripts

A program is made up of one or more scripts, which are collections or **stacks of blocks** that begin with a hat block. Scripts determine how sprites interact with each other and the stage. The CODing Sim activity includes both beginner and advanced levels. The advanced option requires more complex and sophisticated code, encourages higher-level thinking, and is more appropriate for students with experience using block-based programming languages

Choosing the Coding Requirements

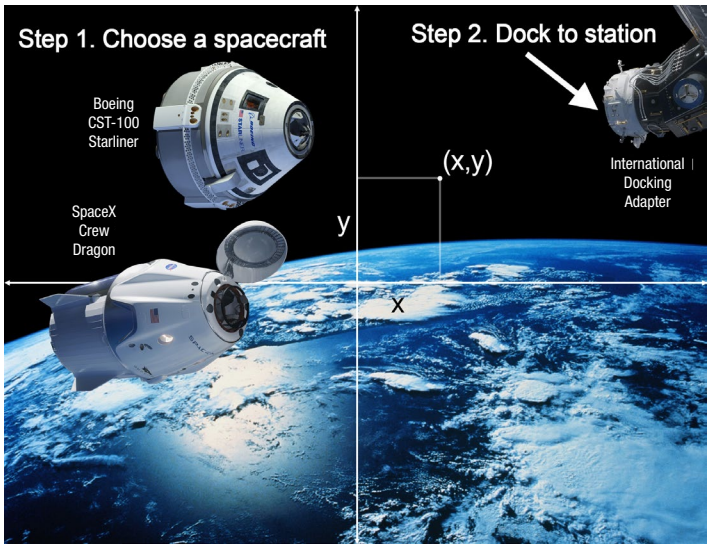


Beginner Coding Requirements

- ▣ Space station IDA sprite is positioned in the top right corner of the stage.
- ▣ Space station IDA sprite is the front layer.
- ▣ When the CCP spacecraft sprite is docked to the space station IDA sprite, it fits within quadrant I of the stage.
- ▣ CCP spacecraft and space station IDA sprites are proportional.
- ▣ The CCP spacecraft sprite starts in quadrants II or III.
- ▣ The CCP spacecraft sprite docks autonomously with the space station IDA.
- ▣ Speed of CCP spacecraft sprite (relative to space station IDA sprite) is no greater than 50 pixels per second.
- ▣ On-screen instructions guide CODing Sim interaction.

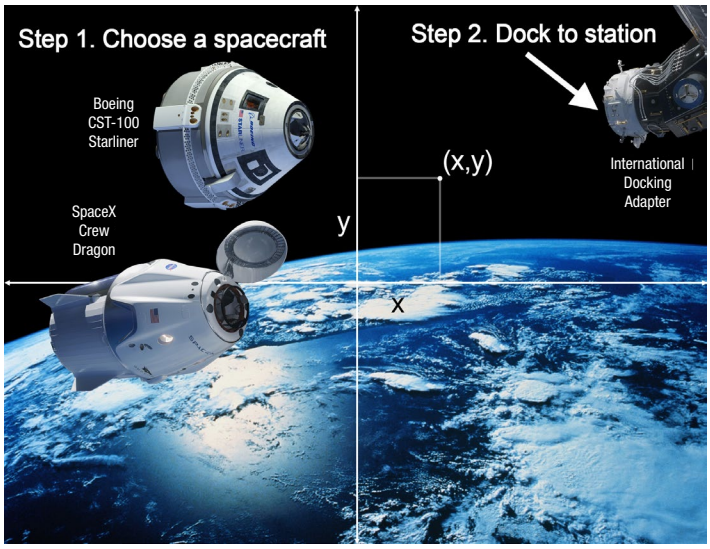
Advanced Coding Requirements

- ▣ Space station IDA sprite is positioned in the top right corner of the stage.
- ▣ Space station IDA sprite is the front layer.
- ▣ When the CCP spacecraft sprite is docked to the space station IDA sprite, it fits within quadrant I of the stage.
- ▣ CCP spacecraft and space station IDA sprites are proportional.
- ▣ The CCP spacecraft sprite starts in quadrants II or III.
- ▣ The CCP spacecraft sprite starts in a random location.
- ▣ User has option to choose CCP spacecraft (Boeing or SpaceX).
- ▣ The CCP spacecraft sprite docks autonomously with the space station IDA.
- ▣ The CCP spacecraft sprite docks to space station IDA using manual controls.
- ▣ User has option to choose between autonomous or manual docking.
- ▣ Speed of CCP spacecraft sprite (relative to space station IDA sprite) is no greater than 50 pixels per second.
- ▣ Manual controls simulate thrusters which move the spacecraft in the opposite direction than the engine fires.
- ▣ Mission success or failure indicated by sound and/or on-screen visual cues.
- ▣ On-screen instructions guide CODing Sim interaction.
- ▣ Program stops when parameters for mission success or failure are met.



Select Coding Requirements

- ☐ Space station IDA sprite is positioned in the top right corner of the stage.
- ☐ Space station IDA sprite is the front layer.
- ☐ When the CCP spacecraft sprite is docked to the space station IDA, it fits within quadrant I of the stage.
- ☐ CCP spacecraft and space station IDA sprites are proportional.
- ☐ The CCP spacecraft sprite starts in quadrants II or III.
- ☐ The CCP spacecraft sprite starts in a random location.
- ☐ User has option to choose CCP spacecraft (Boeing or SpaceX).
- ☐ The CCP spacecraft sprite docks autonomously with the space station IDA.
- ☐ The CCP spacecraft sprite docks to space station IDA using manual controls.
- ☐ User has option to choose between autonomous or manual docking.
- ☐ Speed of CCP spacecraft sprite (relative to space station IDA sprite) is no greater than 50 pixels per second.
- ☐ Manual controls simulate thrusters which move the spacecraft in the opposite direction than the engine fires.
- ☐ Mission success or failure indicated by sound and/or on-screen visual cues.
- ☐ On-screen instructions guide CODing Sim interaction.
- ☐ Program stops when parameters for mission success or failure are met.



1. Create a Scratch or Snap! account

If working in the cloud, students will need to create individual or team accounts at http://scratch.mit.edu or https://snap.berkeley.edu/ where they can sign in and create a new project.

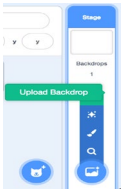
Alternatively, Scratch and Snap! can be downloaded to run locally on a computer. Projects can be imported and exported without an active internet connection.

2. Download all necessary files

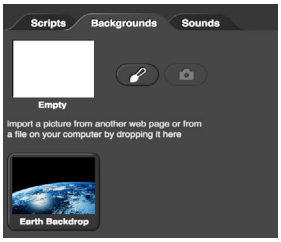
Download and save the image files of the Earth backdrop, CCP spacecraft sprites, and space station IDA sprite.

3. Stage Setup

Select and upload the Earth backdrop image as the background for the stage

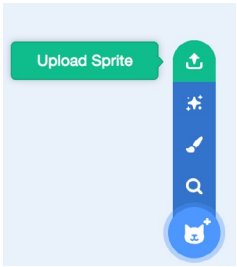


In Scratch, create a new backdrop by clicking **Upload Backdrop** and selecting the downloaded Earth backdrop image that you saved to the computer.

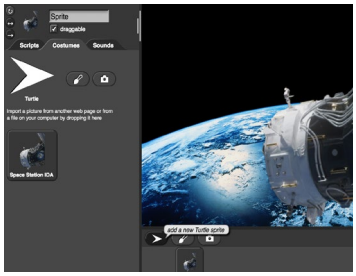


In Snap!, select **Stage** and **Backgrounds**, then drag and drop the downloaded Earth backdrop image just below the Empty background.

4. Sprite Setup



In Scratch, delete the existing cat sprite by selecting it in the **Sprites** window and clicking the x. Add a new sprite by clicking **Upload Sprite**. Select the downloaded space station IDA sprite that you saved to the computer. You will now see the IDA over the Earth background. Repeat for additional sprites.



In Snap!, select **Sprite** and **Costumes**, then drag and drop the space station IDA sprite from your computer to the **Costumes** window just below the **Turtle**. Click "add a new Turtle sprite" below the **Stage** and repeat the steps to upload additional sprites. Right click a sprite to delete it.

Upload each spacecraft image as a separate **costume** for the CCP spacecraft sprite.

Section 5

Coding Tips

Note: Each tip below provides examples of scripts to get you started, but there are many different solutions for each problem. Do not be restricted by a given example.

Jump to:

Tip 1 – Size of Sprites

The sizes of the sprites change when imported into Scratch or Snap! to fit within the stage dimensions. The sizes of both the CCP spacecraft and the space station IDA sprites must be reduced further so they don't overtake the entire stage while still remaining proportional to each other. Explore the **set size to** block in the **Looks** folder to reduce the size of each sprite.

Boeing CST-100 Starliner Sprite

Image dimensions: 980 × 886 pixels

Imported dimensions: 398 x 360 pixels

Space Station IDA Sprite

Image dimensions: 1144 × 1538 pixels

Imported dimensions: 268 x 360 pixels

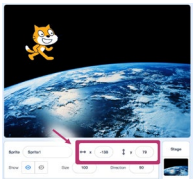
SpaceX Crew Dragon Sprite

Image dimensions: 2719 × 1834 pixels

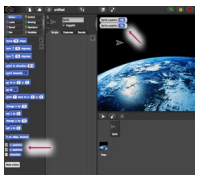
Imported dimensions: 480 x 324 pixels

Tip 2 – Learn about the Stage

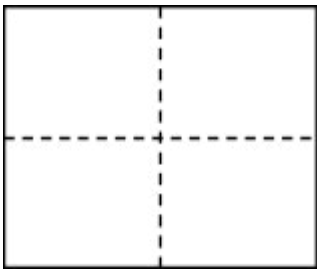
Drag the sprite around the stage to learn how position is defined in the program.



In Scratch, the position of the sprite is located just below the stage.



In Snap!, check the **x position** and **y position** in the **Motion** blocks to make them viewable on the stage.



- Think of the stage like a Cartesian Coordinate system for graphing points.
- Where is the origin?
- What are the x- and y-coordinates at the boundaries of the stage?
- Label the four quadrants counterclockwise starting in the top right (quadrants I, II, III, IV).

Tip 3 – Positioning Sprites

After learning about stage position, explore the **go to x: y:** block in the **Motion** folder to position your sprites. The space station IDA sprite will be stationary and positioned in the top right corner of the stage. The **go to front** block in the **Looks** folder allows the IDA to be in the foreground making the docking simulation look more realistic.

The CCP spacecraft sprite must begin somewhere in quadrants II or III. Beginner coders will select specific (x, y) coordinates as a starting position. Advanced coders can use the random number generator with specific parameters to create a random starting position within those quadrants.

Tip 4 – Sprite Motion

Allow students time to explore the various **Motion** blocks to see what each does. Which motion blocks best simulate how a vehicle moves in space? Which are most realistic for each scenario (automated docking versus manual or controlled docking)? How can we couple the Motion block with a **Control/Events** block to automate or move based on user input? Check out some of the possible solutions below.

Automated Motion

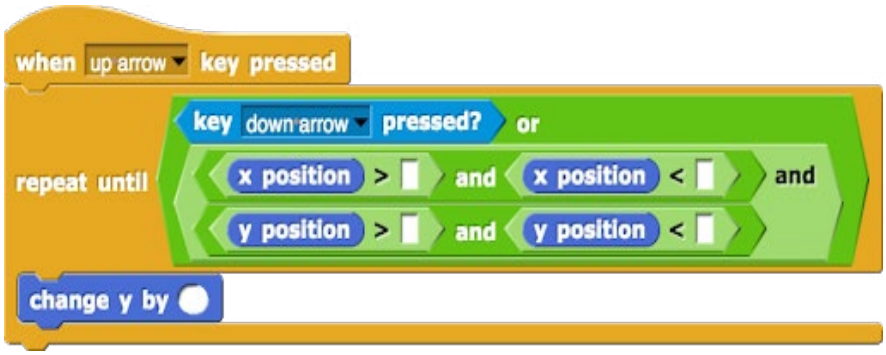
The **glide** block provides a smooth automated motion. How can the speed of the spacecraft be controlled? It must be no greater than 50 pixels/second. Remember that speed is the ratio of distance to time. Click on the **distance to** block to measure the distance between the spacecraft and the IDA sprites. If the distance is known, then the time for the given speed can be calculated. (*Note: students can use the mathematical formula for calculating the distance between two points to check the value of the distance to block.*)

What if the distance varies because the spacecraft is placed randomly in quadrants II or III? Then combine the **ratio**, **distance to**, and **glide blocks** to create a script that calculates the time for any distance. (*Note: the* **glide** *block is looking for an input of seconds, so the ratio should be distance/speed.*)

Manual or Controlled Motion
In the basic simulation, students should not need to turn or rotate the sprites, so all movement is translational in the x- and y-direction. Controlling yaw could be an extension to this activity.

How do you code a sprite to go up or down, left or right? To achieve translational motion, create scripts for motion in each direction using **change x by** or **change y by Motion** blocks. To allow the spacecraft to move more fluidly combine these Motion blocks with **Control/Events** blocks such as **when key pressed** and **repeat until**. The parameters for repeating might be user input such as pressing a key, reaching a certain point on the stage (see example below), or the distance between the CCP spacecraft and space station IDA sprites.

Although the specific numerical values have been deleted, below is an example for a "thruster" that moves the spacecraft sprite "down" when the engine fires "up." How could you modify this script to move the spacecraft sprite to the right?



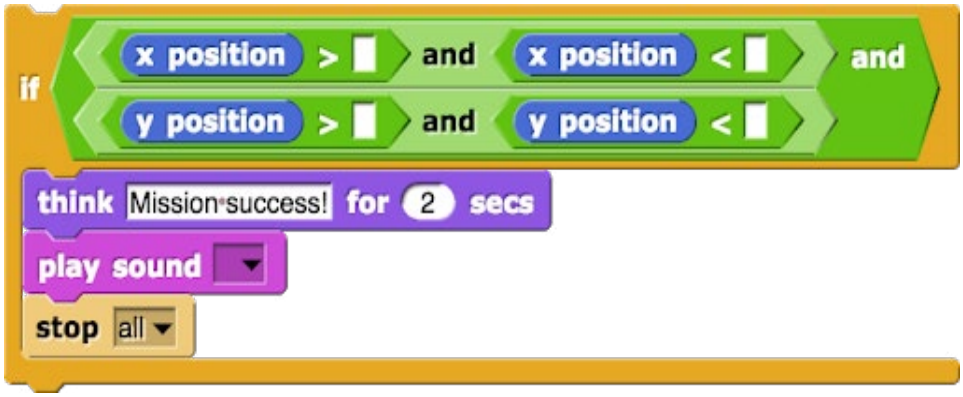
Experimenting with and embedding combinations of Motion, Control/Events, Sensing, and Operators blocks as parameters to define mission success and failure will be the **most challenging and time-consuming** part of this activity. The key is starting simple and adding complexity, testing individual scripts to see how they function, and learning how the blocks interact. Download and import example code from the Materials section into Scratch or Snap! to review the complete code with detailed comments.

Tip 5 – Audio or Visual Cues

Consider adding audio or visual cues to prompt users for input or indicate mission success or failure. In Scratch, sounds can be added to each sprite from the sound library. In both Scratch and Snap!, you may also upload or create your own sounds.

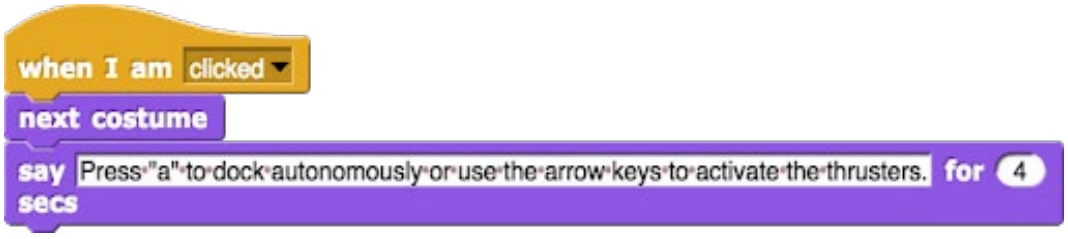
The **say** and **think** blocks in the **Looks** folder allow sprites to send messages. Visual cues might also include text or images uploaded as additional sprites that you **show** or **hide** at various points of the program.

Below is an example of part of the script for the CCP spacecraft sprite announcing when it has successfully docked with the space station IDA sprite through a message and with sound, which play when the sprite is within a specified region designated as “successful docking” by the programmer(s).



Tip 6 – User Interaction

Using visual cues to prompt action paired with the appropriate **Control/Events** blocks, users are able to interact with the simulation and make choices directed by the code. The following example allows the user to click the sprite to choose between different CCP spacecraft and then decide to dock autonomously or manually.



Section 6

Grading Rubric

Rubric Category	Score
Program Execution and Output • Program simulates a spacecraft docking with the space station. • Program meets all of the coding requirements. (X all that apply) ☐ Space station IDA sprite is positioned in the top right corner of the stage. ☐ Space station IDA sprite is the front layer. ☐ When the CCP spacecraft sprite is docked to the space station IDA, it fits within quadrant I of the stage. ☐ CCP spacecraft and space station IDA sprites are proportional. ☐ The CCP spacecraft sprite starts in quadrants II or III. ☐ The CCP spacecraft sprite starts in a random location. ☐ User has option to choose CCP spacecraft (Boeing or SpaceX). ☐ The CCP spacecraft sprite docks autonomously with the space station IDA. ☐ The CCP spacecraft sprite docks to space station IDA using manual controls. ☐ User has option to choose between autonomous or manual docking. ☐ Speed of CCP spacecraft sprite (relative to space station IDA sprite) is no greater than 50 pixels per second. ☐ Manual controls simulate thrusters which move the spacecraft in the opposite direction than the engine fires. ☐ Mission success or failure indicated by sound and/or on-screen visual cues. ☐ On-screen instructions guide CODing Sim interaction. ☐ Program stops when parameters for mission success or failure are met.	
Design of Logic and Standards • Programmer understands and follows the rules of the programming language. • All errors or bugs are eliminated through code. • Program is well organized and design choices are logical and appropriate. • Programmer goes beyond the basics, explores the addition of creative or more in-depth scripts, and demonstrates originality. • Programmer understands and applies STEM concepts.	
Project Management • Time is used constructively to complete project, add additional elements or advanced coding techniques, and collaborate with others effectively. • Programmer implements design process to come up with ideas, choose a solution, build code and test results.	
TOTAL (out of ___ pts possible)	

4 (Advanced) = All criteria (procedures, steps, and details) are met or followed with rare mistakes.
3 (Proficient) = Most criteria are met with only a few mistakes.
2 (Developing) = Many criteria are not met and/or there are many mistakes.
1 (Beginning) = Most criteria are not met.
0 (No effort) = No effort to meet criteria.

Section 7
Additional Resources

Explore Mars with Scratch
https://www.jpl.nasa.gov/edu/teach/activity/explore-mars-with-scratch/

NASA Computer Science Educational Resources
https://www.nasa.gov/audience/foreducators/computer-science-basics.html

Scratch
https://scratch.mit.edu/

Snap!
https://snap.berkeley.edu/

Google CS First
https://csfirst.withgoogle.com/en/home

Code.org
https://code.org/

NASA Audio
https://www.nasa.gov/connect/sounds/index.html

Web sites may provide teachers and students with background information and extensions. Inclusion of a resource does not constitute an endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Section 8
Extension: Create Your Own Sprite

Students may also create their own spacecraft sprite using the paint tool in Scratch or Snap! or another pixel or graphic art software of their choice. Some examples include: Autodesk Sketchbook, Piskel App, Pixie, Pixlr, Gimp, etc. Remember to save the image as a PNG or SVG to maintain a transparent background. Creating a spacecraft that looks three dimensional requires students to exercise linear perspective.