littleBits™

Lesson Plan Sample

TUG OF WAR

Hello!

Thanks for your interest in the littleBits Code Kit! This kit comes with lessons, instructions & resources for 100+ activities, including 10 lessons and 4 main inventions. We wanted to give you a sneak peek at these support materials, so you can picture what your lessons may look like. We're still testing these materials before officially launching the kit in June, so they may change slightly between now and then.

The Code Kit experience was designed with educators and will be broken down into a series of lessons, which will be supported by guided invention builds and concept tutorials in the Code App (covering loops, variables, logic and functions):

LESSON 1: 'Hello World'
LESSON 2: 'Ultimate Shootout'
LESSON 3: 'Hot Potato...of Doom!'
LESSON 4: 'Rockstar Guitar'
LESSON 5: 'Tug of War'
LESSON 6: Boss Level Challenge (Open prompts)
Plus Extension Lessons!

The 'Tug of War' sample on the next page is an intermediate to advanced lesson, meant to be tackled by students and teachers who have completed lessons 1-4, or have a foundation in the above coding concepts. Please note that this is a working DRAFT; content and structure are likely to change.

Each lesson plan will be accessible and customizable on google docs, with links to student-facing presentation slides, invention templates, concept videos and student support tools. A Code Kit educator site will have links to all of these resources.

For launch, a unit plan will provide pacing support for those that may want to dip their toes into a 1 week coding introduction vs. a full 8-10 week curriculum.

We aim to provide a hyper engaging, hands on introduction to coding and design engineering – that is also easy to teach and implement.

We can't wait to see what you and your students create!

LITTLEBITS LESSON

Tug of War

TEACHER NAME: Adam Driggers, MS839
CLASS/PERIOD: Grade 6,

LESSON OVERVIEW/ESSENTIAL QUESTION:

Students create and remix the Tug of War game to explore how functions are used to structure code.

Essential Question: *When and why are functions needed?*

LESSON TAGS

GRADE LEVEL	SUBJECTS	DIFFICULTY	DURATION
Middle School to High School	Math Science Engineering	Advanced	4, 50 minute class periods

PREREQUISITE KNOWLEDGE

Loops
Variables
Functions
“Ultimate Shootout”
“Rock Star Guitar”
“Hot Potato of Doom”

DESCRIPTION

LESSON OUTLINE	INTRO: Warm up exercise and tutorials: introduction to functions CREATE: Students build the ‘Tug of War’ invention in teams of 2-3 students PLAY: Each group tests out their invention. Students explore how their invention and circuit works, plus the coding concepts behind it. Students discuss how the circuit and code work. REMIX: Students will customize and enhance their inventions. Use the *Challenge Matrix to direct students to remixes of various difficulty.* SHARE: Do an ‘Arcade Walk’ of their remixes. Students provide peer to peer feedback then self-assess their own work using their invention logs.
LESSON OBJECTIVES	-Understand the purpose of functions and how they are used in a program. -Organize and structure code using functions.

-Build and test the solution to challenge problems.

ASSESSMENT STRATEGIES

The invention log checklist (invention log pg 18) can be used to assess your students' understanding of the invention cycle, use of the invention log and ability to attain the objectives of the lesson. For formative assessment while students work, you can use this checklist to ask questions about their current task and ensure that they are on the right track. The checklist can also be used as a self-assessment tool by students as they move from phase to phase. For summative assessment, you can use this checklist to review students' entries into their invention log and assess their understanding of the challenge and the invention process as a whole.

STANDARDS

3-5-ETS1-3
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.

MS-ETS1-3
Analyze data from tests to determine similarities and differences among several design solutions to identify the best characteristics of each that can be combined into a new solution to better meet the criteria for success.

CSTA 1B-A-5-5
Use mathematical operations to change a value stored in a variable.

CSTA 2-A-5-7
Create variables that represent different types of data and manipulate their values

CSTA 2-A-7-4
Interpret the flow of execution of algorithms and predict their outcomes.

CSTA 2-A-4-8
Define and use procedures that hide the complexity of a task and can be reused to solve similar tasks.

VOCABULARY

Function
Variable
Loops
Encapsulate
Procedure

SUPPLIES

BITS	ACCESSORIES	OTHER MATERIALS	TOOLS USED
Code Kit (1 kit per group of 2-3 students)		Laptop with Code app downloaded	Scissors

RESOURCES

ATTACHMENTS	Remix Challenge Matrix for Remix Rubber Duck Card for Play Discussion Comment Coins for Share
TIPS & TRICKS	Headphones are recommended if adding the speaker to an invention.
PACING	DAY 1: CODE TUTORIALS Prep + setup Intro (10 mins) Warm up - code tutorials (20 mins) Share (15 mins) Close (5 mins) DAY 2: CREATE AND PLAY Prep + setup Intro (5 mins) Create (20 mins) Play (20 mins) Debrief/Close (5 mins) DAY 3: REMIX Prep + setup Intro (5 mins) Remix (30 mins) Share (10 mins) Close (5 mins) DAY 4: SHARE Prep + setup Intro (5 mins) Remix Iterations (20 min) Share (20 mins) Close (5 mins)

INSTRUCTIONAL STEPS

STEP 1: SETUP
Duration:

This lesson can be done individually or in small groups (2-3 students).

Each group will need at least one Code Kit, computer, plus one invention log and assessment checklist per student.

Set up a central location in the classroom for assorted materials and tools.

During the create phase, students will construct their first prototypes according to instructions in the Code App. You may want to construct your own example prototype before the lesson begins. Seeing a working model of what they are building can help the students understand the goal of their create phase and will allow you to quickly demonstrate it working in the Play phase.

If you'd like to provide your students with the templates for the soccer goal, be sure to print copies beforehand

NOTES

[notes]

STEP 2: INTRODUCE

Duration: 50 minutes

Review your community code and littleBits basics.

Open up your 'Tug of War' student presentation (link) and review the goals for the day.

Building Background Knowledge (10 minutes)
Break students in teams of 2 or 3
Use this activity to identify prior knowledge and identify misconceptions regarding functions.

Round 1
Give each group a poster paper or blank sheet of paper. Have them write "Function" in the middle of the page and draw a small circle around the word. Ask students to write anything and everything they know about functions.

Note: This activity could also be done digitally. It works well to use a 3 column chart for each round.

Round 2
Have students draw a circle around the outside of their responses from round 1. Watch the functions coding concept video (link). Have students record any new information regarding functions to the chart.

Round 3
Students draw another circle around their round 2 responses. For round 3, students will work on one of the code tutorials for function in the Code App (link). While they are working on the tutorials, they should add any new understandings or ideas about functions to their charts.

Code Tutorials (30 mins)
Assign each of group to a computer workstation. Have students open up the littleBits Code App and click on the "functions" challenge. Students will follow the tutorial and add any new information about functions to their posters.

For remediation/support, hand out or provide link to the 'debugging checklist' (link)

If students are having difficulties with the functions tutorial, they may need to go back and refresh on the other Code Kit tutorials.

Discuss: Give One, Get One, Move On (10 mins)

Use this procedure to spread learnings and to see what "stuck" with participants. You can structure it with movement, or make it a silent, written experience.

Procedure

1. Ask students to individually write down 3-5 key learnings or important ideas about functions. Have people write each idea on a different index card or sticky-note to give away to his or her partners. Students can refer to their circle chart for ideas.
2. Invite the students to get up and mingle with each other.
3. After about 30 seconds, call out "GIVE ONE to a partner."
4. Participants form pairs and each "gives" one of his or her key learnings or important ideas about the topic to the other, so each person "gives one" and "gets one." Time may range from 1-3 minutes.
5. Call out "MOVE ON" and students mingle again.
6. Repeat the sharing for as many ideas as people have to share.

To close, have each student share their favorite new idea or concept from the day. Collect the notes to see what the students took away from this lesson.

NOTES

If students have a strong grasp of loops, variables, and functions, you may want to skip the tutorials and start by creating the Tug of War game.

STEP 3: CREATE

Duration: 20 minutes

Play the 'Tug of War' video to the class (link to presentation).

If using the Invention Log, define criteria for success and constraints that are appropriate for your students. For example, your criteria for success could be that the circuit must contain power, input and output.

At each group workstation, ask students to open up the littleBits Code app and click on the 'Tug of War' invention. Students will watch the instructional video in their groups to create and code their invention.

Encourage students to reference the Bit Index (pg 7-27 in their invention guides) if they get stuck or want to learn more about a particular bit or accessory. For younger students, you may want to pause the class after each step to troubleshoot any common problems, as well as share successful build strategies amongst the groups.

NOTES

You may prefer to have a digital copy of your Invention Log (link). This will enable students to add screenshots, pics and videos into their documentation.

View classroom management tips (link) for support with team work and troubleshooting.

littleBits education

STEP 4: PLAY

Duration: 10 minutes of play + 10 minutes for discussion

As you move through the play prompts, be sure to have students record their PLAY process and reflections in the Invention Log (starting with "how did you testing go?").

How did your testing go?: Once the inventions have been constructed, students should test their prototypes to make sure it works and to explore the circuit functionality.

Students should test the tug of war game. If the circuit doesn't work:
-check the circuit; make sure there are blue powersnaps on one end on the button
-make sure the button is pressed all the way down
-make sure your power bit is switched on and the cable connections are secure.
-check for low batteries

Rubber Ducking Discussion Protocol
Rubber Ducking is a debugging practice where a programmer will articulate a problem aloud, historically to a yellow rubber duck. The act of speaking the problem out loud can often lead the programmer to a solution. After students have had a chance to play with the circuit, ask them to "rubber duck" the circuit and the program to their partner.

Setup (2 minutes)
- If students are working in pairs have them decide who will discuss the circuit and who will discuss the program. If working in small groups have students take turns or work together to discuss each part.
- Allow students to use the rubber duck card visual to identify who is speaking and who is listening.

Round 1 (3 minutes)
- One student describes how the circuit works to their partner/group
- Have the prompts displayed on the board or on a handout to help students who are struggling to get started.
- Circuit Prompts:
    - "Walk through the circuit from beginning to end, describing what is happening at each bit."
    - "Describe the inputs"
    - "Describe what happens when the input is triggered"
    - "Describe the outputs"
    - "Describe what the Code Bit is doing"

Round 2 (3 minutes)
- One student describes how the code works to their partner/group
- Have the prompts displayed on the board or on a handout to help students who are struggling to get started.
- Code Prompts:
    - "Walk through the code from beginning to end, describing what is happening at each block."
    - "Describe the variables"
    - "Describe how the loops are working"
    - "Describe what happens when an input is triggered."
    - "Describe what the "move box" function is doing"

Debrief (2 minutes)

- Have students share one interesting thought they heard when they were the rubber duck.
- Sentence Starter: "When I was the rubber duck I heard..."

Be sure to have students record their notes and processes in the Invention Log.

NOTES

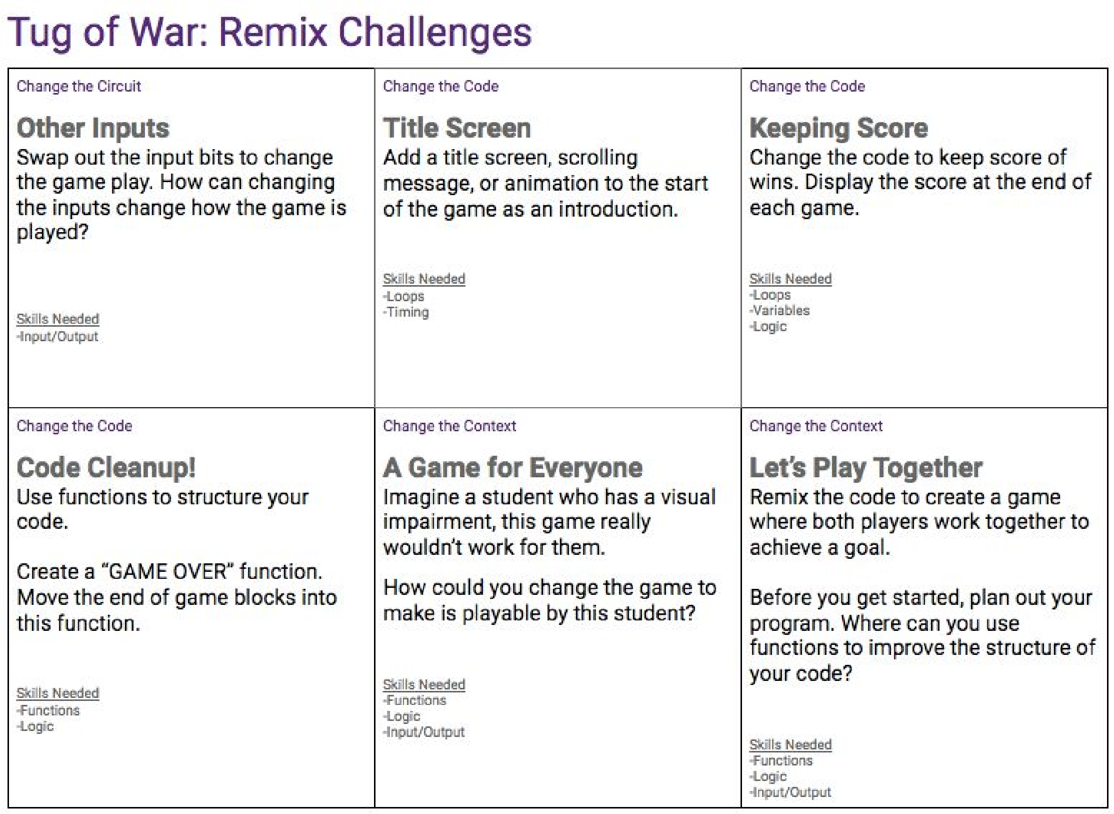
For support during the play phase, reference the invention advisor tips

STEP 5: REMIX

Duration: 30 minutes

There are many different entry points for remixing Tug of War, from swapping out some bits, changing some code, or completely changing the context of the game.

Use the Challenge Matrix to allow students to self pace through the remix phase. You can either designate a starting challenge or have students choose their challenge level.



Tug of War: Remix Challenges

Change the Circuit **Other Inputs** Swap out the input bits to change the game play. How can changing the inputs change how the game is played? Skills Needed -Input/Output	Change the Code **Title Screen** Add a title screen, scrolling message, or animation to the start of the game as an introduction. Skills Needed -Loops -Timing	Change the Code **Keeping Score** Change the code to keep score of wins. Display the score at the end of each game. Skills Needed -Loops -Variables -Logic
Change the Code **Code Cleanup!** Use functions to structure your code. Create a "GAME OVER" function. Move the end of game blocks into this function. Skills Needed -Functions -Logic	Change the Context **A Game for Everyone** Imagine a student who has a visual impairment, this game really wouldn't work for them. How could you change the game to make is playable by this student? Skills Needed -Functions -Logic -Input/Output	Change the Context **Let's Play Together** Remix the code to create a game where both players work together to achieve a goal. Before you get started, plan out your program. Where can you use functions to improve the structure of your code? Skills Needed -Functions -Logic -Input/Output

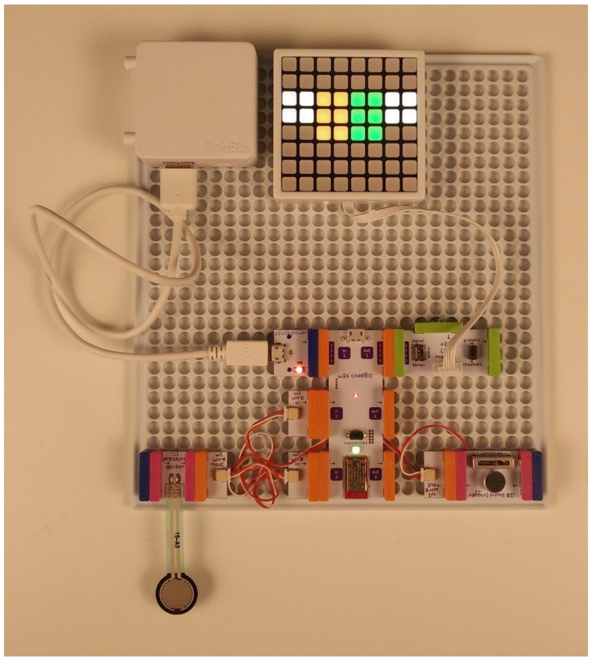
When students feel like they have completed each remix challenge have them submit their solutions either digitally or by having you look over their code. Giving a check or stamp in the box of the completed challenge can act as extra motivation for students to stay focused on remixing. You may want to collect the Challenge Matrix for each group as formative assessment data.

Below is a description and possible solution for each challenge level:

Change The Circuit

Other Inputs
Prompt:
Swap out the input bits to change the game play. How can changing the inputs change how the game is played?

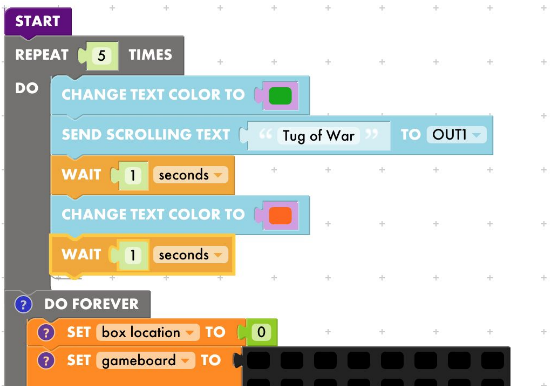
Students will discover that the buttons can be replaced with other input bits. A possible solution is to change one button to the Sound Trigger (i20) bit and the other button to the Pressure Sensor (i11) bit. Many different inputs could work, so if you have any other bits around get them out and let kids experiment.



<u>**Change the Code**</u>
Title Screen
Prompt:
Add a title screen, scrolling message, or animation to the start of the game as an introduction.

Lets add a title screen to the game using scrolling text.

<u>Possible Solution</u>



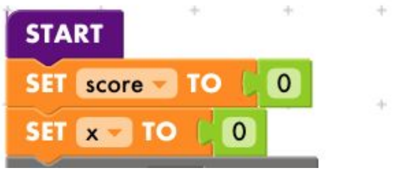
```
START
REPEAT 5 TIMES
DO  CHANGE TEXT COLOR TO 🟩
    SEND SCROLLING TEXT " Tug of War " TO OUT1
    WAIT 1 seconds
    CHANGE TEXT COLOR TO 🟧
    WAIT 1 seconds
DO FOREVER
  SET box location TO 0
  SET gameboard TO
```

In the example above, the scrolling text will show the title of the game for 10 seconds before the game starts.

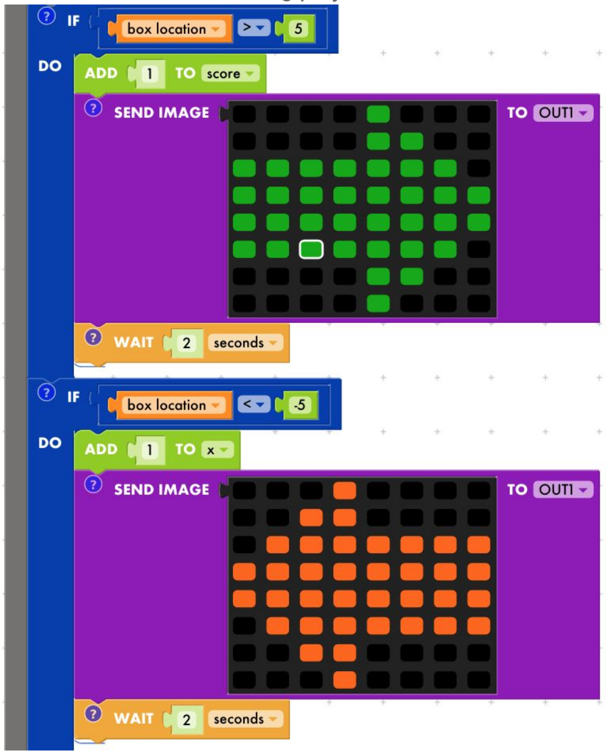
Keeping Score

Prompt:
Change the code to keep score of wins. Display the score at the end of each game.

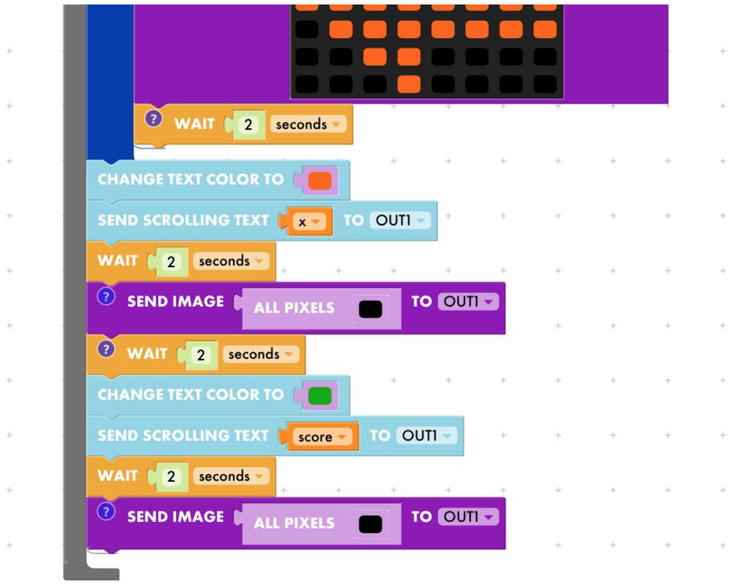
<u>Possible Solution</u>
At the start of the game, set the variable for orange and green to 0



When there is a win, update the score of the winning player.



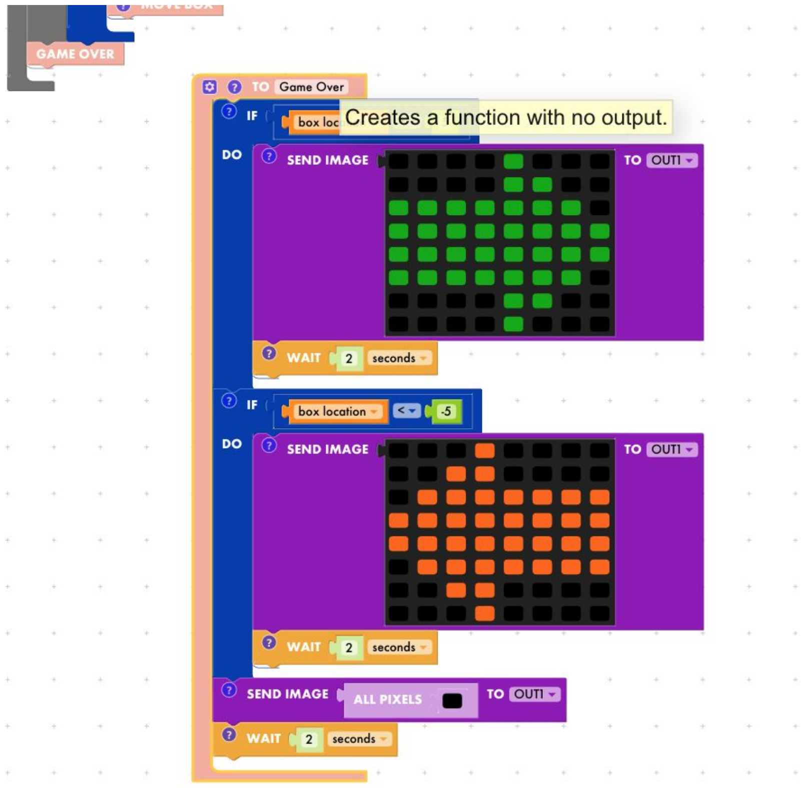
What good is keeping score if we don't ever show the user? Let's display the score at the end of the game.



```
WAIT 2 seconds
CHANGE TEXT COLOR TO
SEND SCROLLING TEXT x TO OUT1
WAIT 2 seconds
SEND IMAGE ALL PIXELS TO OUT1
WAIT 2 seconds
CHANGE TEXT COLOR TO
SEND SCROLLING TEXT score TO OUT1
WAIT 2 seconds
SEND IMAGE ALL PIXELS TO OUT1
```

Code Cleanup
Prompt:
Use functions to structure your code. Create a “GAME OVER” function. Move the end of game blocks into this function.

One of the benefits of using functions is that they can be used to make the code more easily read by other coders. Functions can separate code into smaller, more understandable chunks. This is often called “encapsulation.” Try creating a “Game Over” function that separates the end code from the rest of the loop.



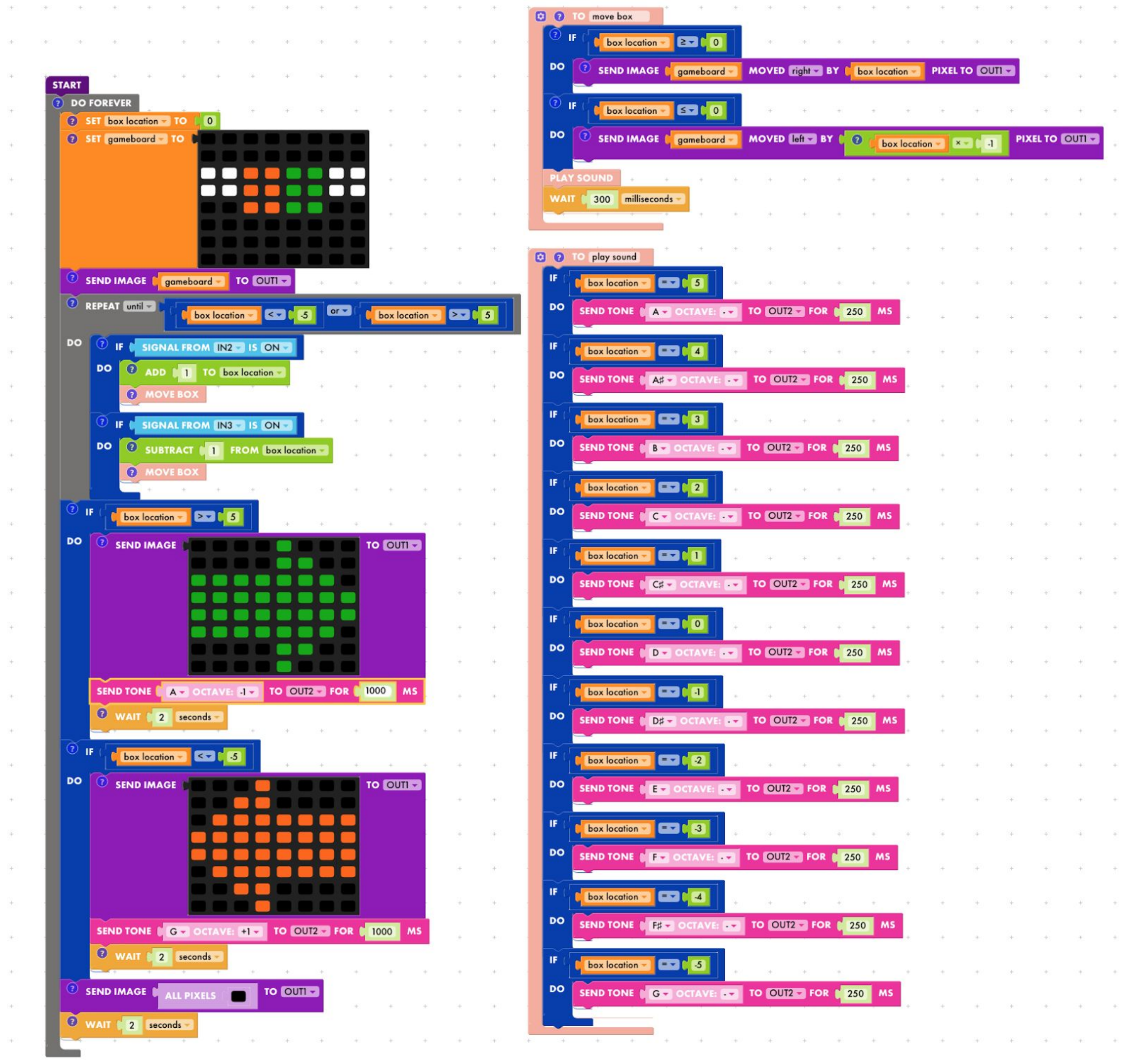
```
GAME OVER

TO Game Over
  IF box loc
  DO SEND IMAGE TO OUT1
     WAIT 2 seconds
  IF box location < -5
  DO SEND IMAGE TO OUT1
     WAIT 2 seconds
  SEND IMAGE ALL PIXELS TO OUT1
  WAIT 2 seconds
```

Creates a function with no output.

<u>**Change the Context**</u>
This level of remix asks students to change both the circuits and the code.

A Game for Everyone
Prompt:
Imagine a student who has a visual impairment, this game really wouldn't work for them. How could you change the game to make is playable by this student?

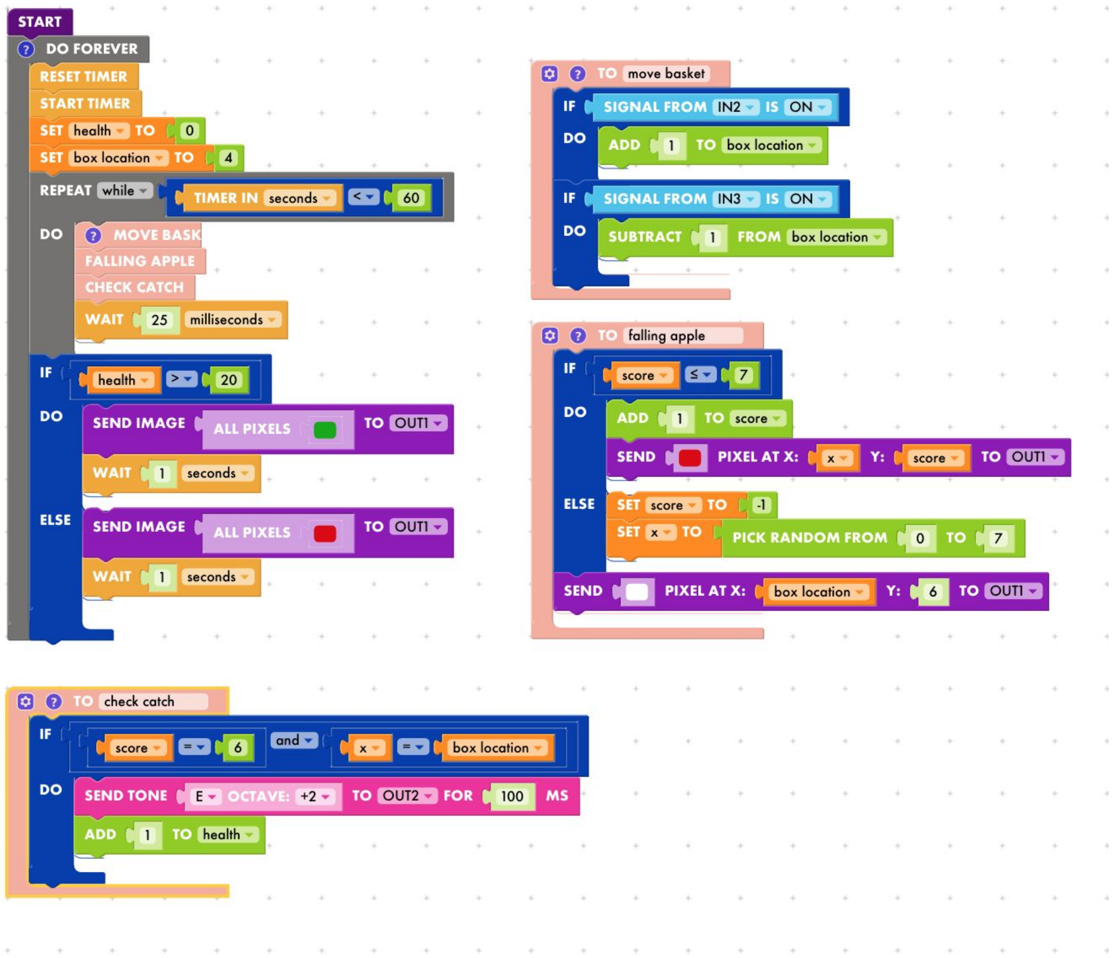
<u>Possible Solution</u>
Example Below uses the speaker as an output, wrapped in a function.



```
START
DO FOREVER
    SET box location TO 0
    SET gameboard TO [image]
    SEND IMAGE gameboard TO OUT1
    REPEAT until  box location < -5  or  box location > 5
    DO
        IF SIGNAL FROM IN2 IS ON
        DO  ADD 1 TO box location
            MOVE BOX
        IF SIGNAL FROM IN3 IS ON
        DO  SUBTRACT 1 FROM box location
            MOVE BOX
    IF box location > 5
    DO  SEND IMAGE [image] TO OUT1
        SEND TONE A OCTAVE: -1 TO OUT2 FOR 1000 MS
        WAIT 2 seconds
    IF box location < -5
    DO  SEND IMAGE [image] TO OUT1
        SEND TONE G OCTAVE: +1 TO OUT2 FOR 1000 MS
        WAIT 2 seconds
    SEND IMAGE ALL PIXELS TO OUT1
    WAIT 2 seconds

TO move box
    IF box location ≥ 0
    DO  SEND IMAGE gameboard MOVED right BY box location PIXEL TO OUT1
    IF box location ≤ 0
    DO  SEND IMAGE gameboard MOVED left BY box location × -1 PIXEL TO OUT1
    PLAY SOUND
    WAIT 300 milliseconds

TO play sound
    IF box location = 5
    DO  SEND TONE A OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = 4
    DO  SEND TONE A♯ OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = 3
    DO  SEND TONE B OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = 2
    DO  SEND TONE C OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = 1
    DO  SEND TONE C♯ OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = 0
    DO  SEND TONE D OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = -1
    DO  SEND TONE D♯ OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = -2
    DO  SEND TONE E OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = -3
    DO  SEND TONE F OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = -4
    DO  SEND TONE F♯ OCTAVE: - TO OUT2 FOR 250 MS
    IF box location = -5
    DO  SEND TONE G OCTAVE: - TO OUT2 FOR 250 MS
```

Let's Play Together
Prompt:
Remix the code to create a game where both players work together to achieve a goal. Before you get started, plan out your program. Where can you use functions to improve the structure of your code?

<u>Possible Solution</u>
The example below is a game called Newton's Apple. The players must use the buttons to move the basket (white pixel) to catch the apple (red pixel). The goal is to catch 20 apples in 1 minute.



```
START
DO FOREVER
    RESET TIMER
    START TIMER
    SET health TO 0
    SET box location TO 4
    REPEAT while TIMER IN seconds < 60
    DO  MOVE BASK
        FALLING APPLE
        CHECK CATCH
        WAIT 25 milliseconds
    IF health > 20
    DO  SEND IMAGE ALL PIXELS 🟩 TO OUT1
        WAIT 1 seconds
    ELSE
        SEND IMAGE ALL PIXELS 🟥 TO OUT1
        WAIT 1 seconds

TO move basket
    IF SIGNAL FROM IN2 IS ON
    DO  ADD 1 TO box location
    IF SIGNAL FROM IN3 IS ON
    DO  SUBTRACT 1 FROM box location

TO falling apple
    IF score ≤ 7
    DO  ADD 1 TO score
        SEND 🟥 PIXEL AT X: x Y: score TO OUT1
    ELSE
        SET score TO -1
        SET x TO PICK RANDOM FROM 0 TO 7
    SEND ⬜ PIXEL AT X: box location Y: 6 TO OUT1

TO check catch
    IF score = 6 and x = box location
    DO  SEND TONE E OCTAVE: +2 TO OUT2 FOR 100 MS
        ADD 1 TO health
```

NOTES

[notes]

STEP 6: SHARE

Duration: 10 minutes

Have students spend a few minutes documenting (video/images) their invention and save any files/screenshots in their invention log (or the file management system of your choice).

Arcade Walk

Set-up a "Arcade Walk" of the students remixes. Remind the students of norms for giving feedback—be kind, be specific, and be helpful. Ask students to look specifically at where and when functions were used in the remixed programs.

Setup (2 minutes)

- Have students place their remixed project and their computer with the code open next to each other.
- Students give feedback on the "comment coins," give each student a few to start and have a pile in case they need more.
- Go over feedback types and have a slide up during the "Arcade Walk."

- Warm Feedback: Something that was done well or is really successful.
    - Cool Feedback: Something that needs improvement. Something that was not so successful. A bug. A confusion.
- Remind students that the purpose is to get and give quality feedback.

Gallery Walk (3 minutes)

- Have students experience each other's projects.
- Students give Warm and Cool feedback using the coins.

Discussion / Reflection (5 minutes)

- Ask groups to review comments and choose one piece of feedback to respond to.
- Do a go around where each group reads the feedback and then gives a response.

Have students record feedback in their Invention Logs.

NOTES

STEP 7: CLOSE

Duration:5 minutes

Have students complete an "exit slip" to determine their understanding of functions. This could be recorded in a notebook, an online submission, or a slip of paper that students turn in at the end of class.

Sample prompts:

- How are functions used?
- Give an example of when to use a function

Collect the feedback forms from each group.

If you are assessing student work, the self-assessment checklist may be handed and filed out (link).

Printable/editable certificates can be used to celebrate your student's achievements (link)

Students should take apart their inventions and put away the Bits according to the diagram on the back of the Bit Index. Students should clean up their workspace and close out their app.

NOTES

none

STEP 8: EXTENSIONS

Duration: [insert time here]

[taking it further ---> coming soon!]