NIM Game

Teacher's Guide

Lesson Overview

In this lesson, students will learn how to build a classic NIM game. NIM is a simple mathematical game for two players that involves a number of markers laid out in a row. Players take turns removing one, two, or three markers with the goal of forcing the other player to take the last marker. This game involves simple logic and the winning approach changes based on whether you are the first or second player.

Grade Level(s)

5-12

Duration

2-3 hours

Objectives

1. Students will be introduced to electronic games, and the idea of "game theory", which in this case is just that it matters who goes first.
2. Students will understand that games like this can be viewed in a way that they can roughly understand that "1" is the same losing position as "4", and in turn, "7" and "10". Thus, whoever moves first can turn off one light, leaving the other player with 10, which can be an equivalent losing position to the last move where only one light remains turned on.
3. Students will be introduced to the idea of "computational thinking", where what that means is creating a set of instructions that either a human or a computer could follow to accomplish a task. Computational thinking as a process can be considered to have 3 phases:
    a. Abstraction: Problem formulation. Describe the task to be done, or problem to be solved.
    b. Automation: Solution expression. Create a set of instructions that a person or a computer can repeatedly carry out to accomplish the task or solve the problem.
    c. Analyses: Solution execution and evaluation. Have the person or the computer do the instructions, and decide if it worked. If not, try again with changes until a successful outcome is achieved.

Supplies

Required

NIM game overlay
2 ribbon cables for LEDs
1 ribbon cable for touchpins
7 touchpoints & backs

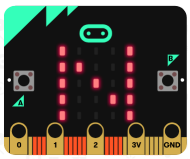
Optional

Masking, painter's or transparent tape

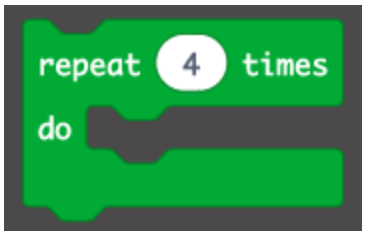
Vocabulary

Debugging
Algorithms
Variables
Conditionals
Compound math statements

MakeCode Blocks Introduced



(compound math operations)



(loop - repeat n times)

Instructional Steps

Warm Up

Time: 30 minutes

Discuss with students the idea of computational thinking. Show a working model of the NIM game, and ask them to write, or create on the whiteboard, a natural language description of how to play the game. Then ask what the code that represents that same game play might look like.

Part of that process could be to look at a diagram of the NIM game box, and to think about which parts might be good to control with a variable, and which parts might use a conditional or event block.

Main Activity

Time: 2+ (as many as 10) class periods depending on the teacher's objectives, students' age, motivation and engagement.

Project Instructions:

Part 1: Assembly

Part 2: Coding

Part 3: Challenges

Teaching Tip: This lesson requires more class discussion and leading through the concepts than some of the other lessons, so it's necessary both to prepare for the discussions and to allow time in class for the students to explore and develop their mental models for the game and the solutions.

The first "Simplest NIM game" program in the instructions is just turning off an LED each time touch sensor T5 is touched. There is no error-checking, or checking for who wins, or even different touch sensors for different players or turning off 1, 2 or 3 LEDs.

This represents what might be the students' (as a group) first proposed program, or they may, based on their comfort level with programming from having done the previous projects, be able to propose a program that uses more touch sensors, and uses if-then conditional blocks to determine if there is a winner.

These possibilities are where the skill of the teacher will help find the best balance of challenge and motivation + skills on the part of the students.

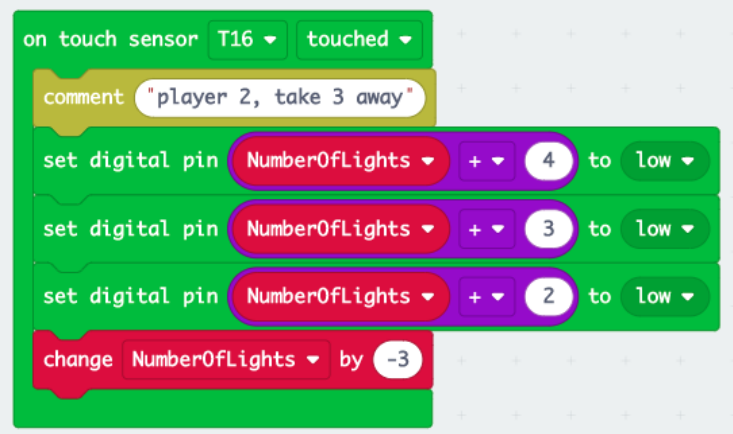
Trouble-shooting Tips

Have a totally functional model available. A fast way to diagnose a problem in a student model is to plug in a micro:bit that is pre-programmed with the NIM program into the MakerBit of a student model. If the model then works, then any errors seen with the LEDs or touch sensors is a coding error in the student model. If the physical model does not work (touch sensors don't respond, LEDs don't turn on, LCD doesn't display text.) then there is most likely a wiring error in the student model.

Challenges

Time: 4-5 class periods

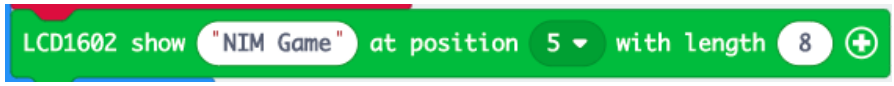
1. For the “on touch” event groups for taking away 3 LEDs, do you think this code would work for “taking away 3”?



   Which way of writing the code do you prefer? Why? Are there any advantages and disadvantages to each way of doing the code?
   A: Pro: Using the 3 blocks that turn the LEDs off, and then subtracting 3 from NumberOfLights uses fewer blocks, so the code is shorter. **Con**: Mentally, it takes a little more effort to “de-code”, in your head, what is happening with +4, +3, and +2 for the pin numbers of the LEDs. For the following challenges, though, the solution programs will use the more compact form, as shown above.
   https://makecode.microbit.org/#pub:_1EjJ77H6rD02
2. What happens if there are two LEDs still on, but a player chooses to take away 2 or 3, rather than 1, which would be the usual winning move? Will your program turn off all the LEDs? Is this still a winning move? Is it a program bug? How do you think the program should behave, and can you add to your code to make it work that way?

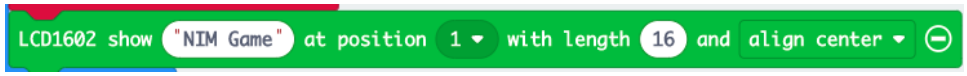
A: https://makecode.microbit.org/#pub:_Vjc7P5arePoK

3. Can you print "NIM Game" on the first row of the LCD and center it?
   A1: (16-(length of text))/2 +1 = starting position.



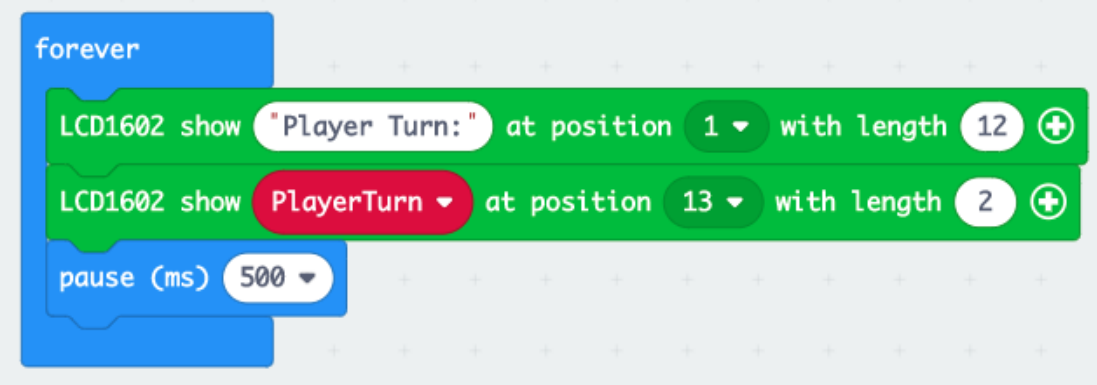
https://makecode.microbit.org/#pub:_AyH6LRTEzhmW

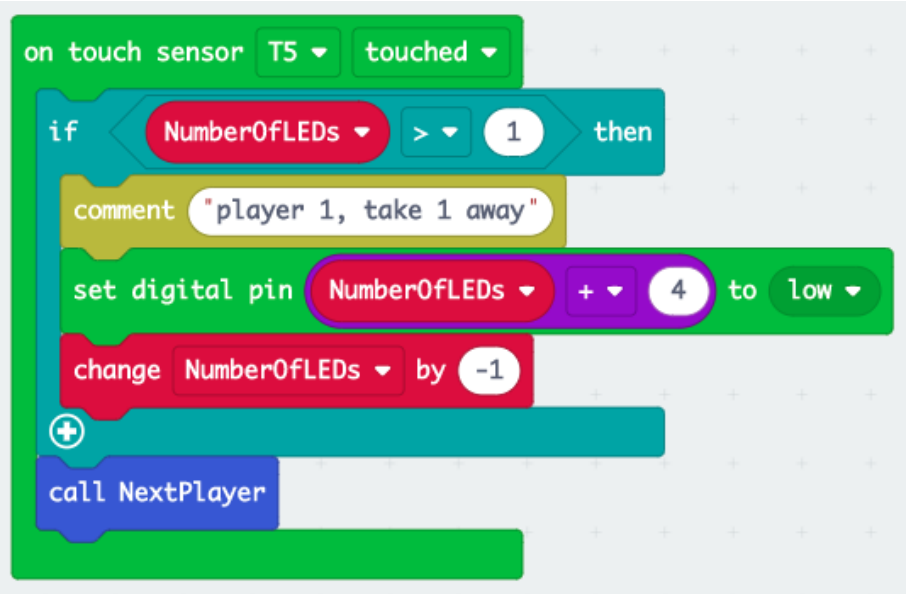
A2:

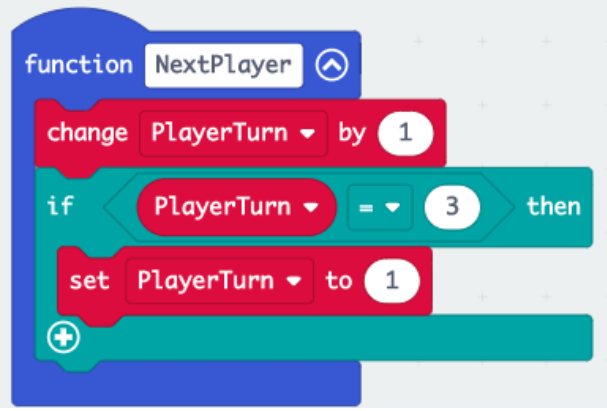


https://makecode.microbit.org/#pub:_iADTHyXb4E0W

4. Can you indicate whose turn it is ("player 1" or "player 2") on the LCD? Hint: Create a new variable called "PlayerTurn", and set it to "1". As part of each on-touch event, add 1 to the variable ("change PlayerTurn by 1"). Add a test for when PlayerTurn = 3 to then set it back to 1. So that you don't have to write the same code six times, make a function called "NextPlayer", and call that from each of the on-touch events.

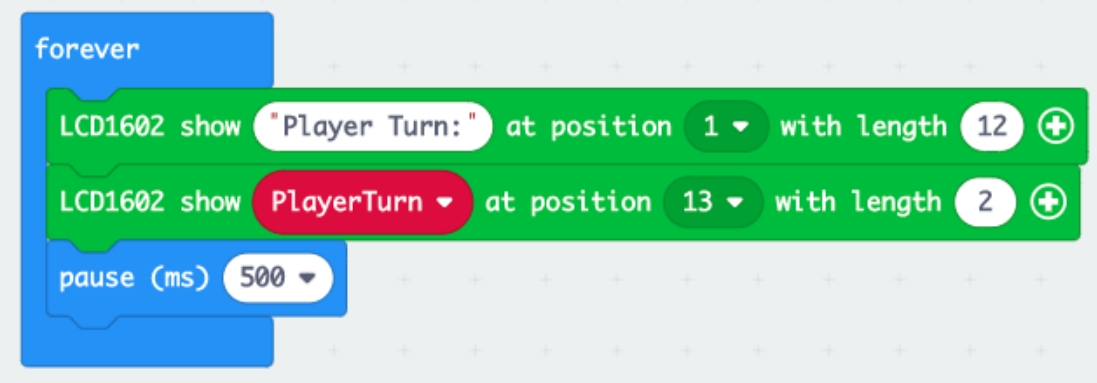






A: https://makecode.microbit.org/#pub:_HaL4KMhtVJhy

5. **Joining Text**
Look at this program code:

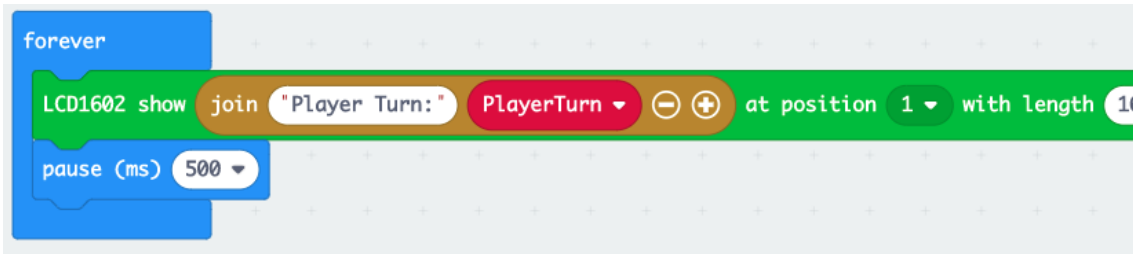


And notice that it is printing "Player Turn:" and the variable PlayerTurn together on the same line of the LCD. For the second block, you had to count how many characters are in the phrase "Player Turn:" (12), and then add one for the next

position for the number value of PlayerTurn.

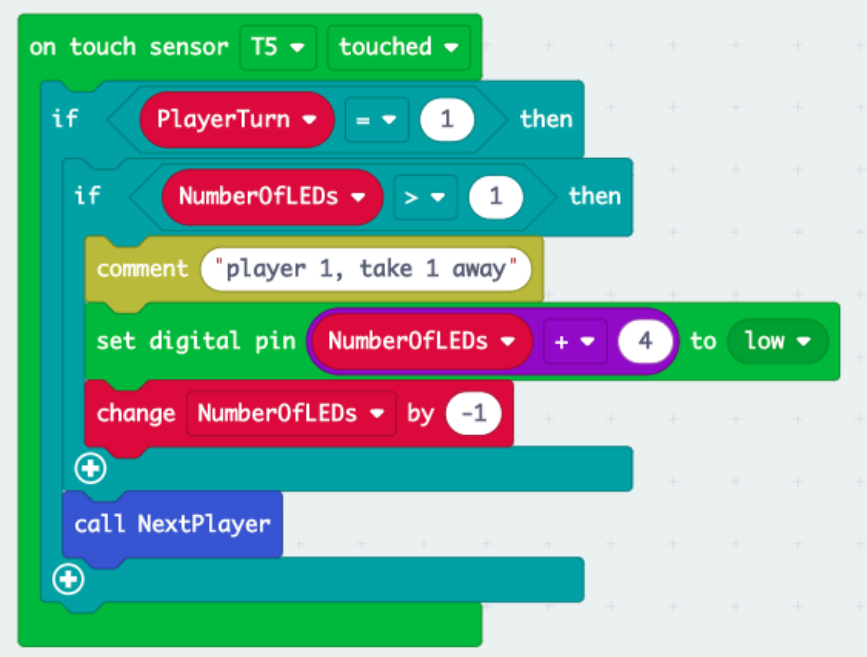
There is a way for the computer to help you with all this, and that is the “join” block that you can find under Advanced, and Text:

The join block does just what it says: It puts any number of elements together in the same string of text, so now your code can look like this:



Try adding the join block to your code now to replace the two LCD1602 show blocks, so that now there will just be one.
A: https://makecode.microbit.org/#pub:_3tddjKdwVD03

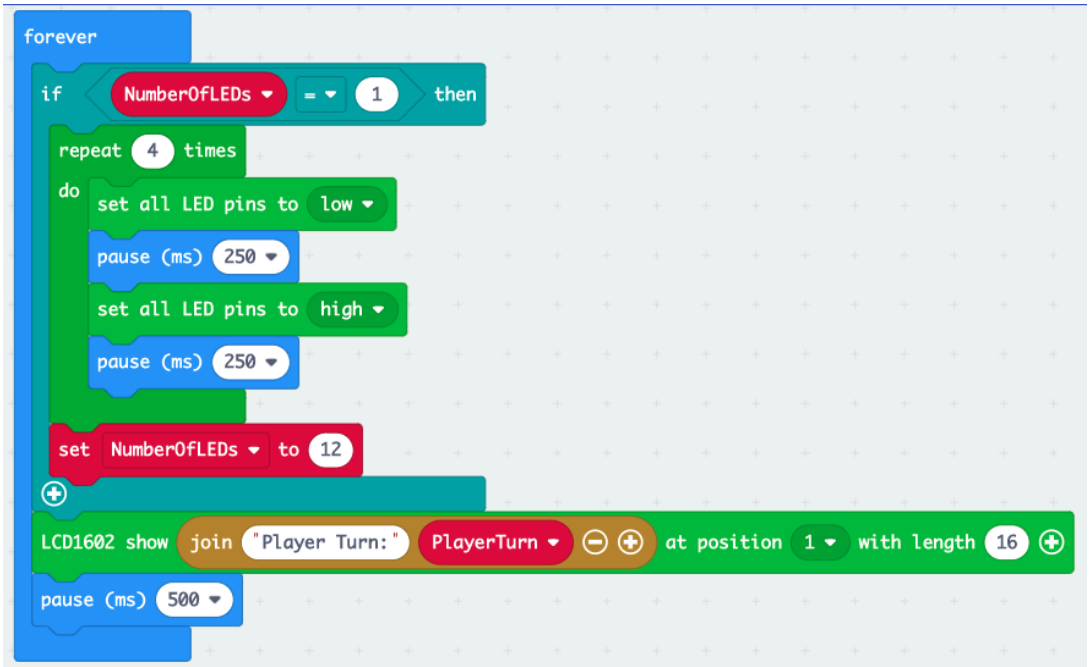
6. Is there a way to only allow the player whose turn it is to touch their 1,2,3 buttons, and not accept inputs from the “other” player on each turn? **Hint:** Use an if-then test for the PlayerTurn at the beginning of each on-touch event block, and put the code that was already there inside the PlayerTurn test block.
The code will then look like this:



A: https://makecode.microbit.org/#pub:_J7wPEUHvdgaU

7. Can you modify the code to blink all the LEDs 4 times after "Start Over" (T10) is touched, and end with all the LEDs on?
   Hint: Try using the "repeat (4) times" block.
   Does it matter whether you blink with set high, then set low, compared to set low, then set high? Can you explain to someone, or in a Google doc, the difference between these two samples of the start over code?
   A1: https://makecode.microbit.org/#pub:_Dk9PDuX87ix3 (high/low)
   A2: https://makecode.microbit.org/#pub:_bdyDwJ9agY59 (low/high)
   A3: If the LEDs are made to blink in the order of high, then low, they end the blink sequence in the low ("off") state, and so an extra block is required to turn all the LEDs on again to start another game. If they blink in the order of low, then high, they end the blink sequence in the high ("on") state, and so everything is ready to start over without needing the extra block to set all LEDs high.

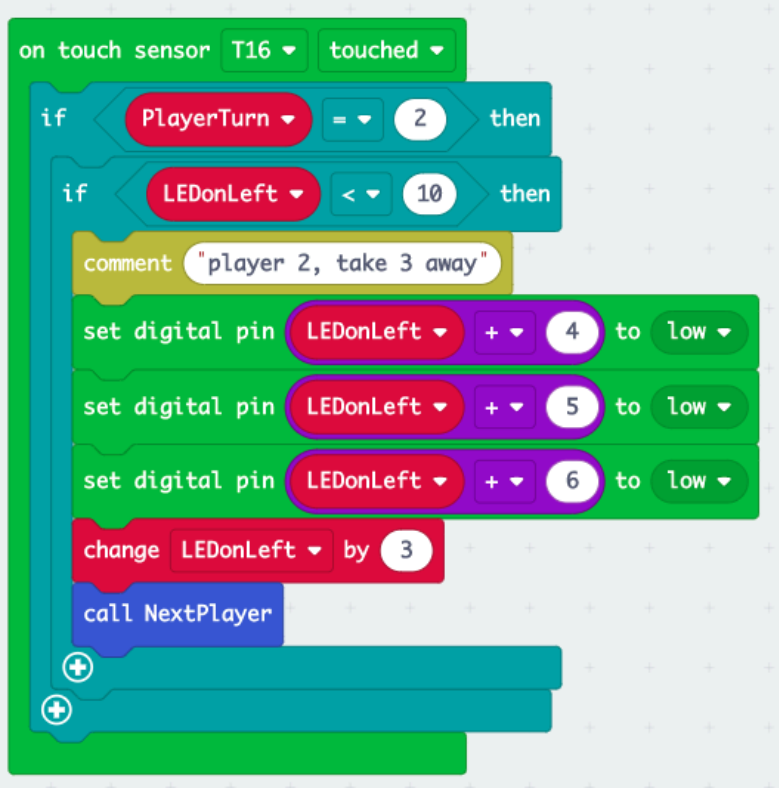
8. In these previous versions of the program, the players have to decide when the game is won. Can you add some program blocks to flash the LEDs when the last LED is left on, and then re-start the game? **Hint**: In the forever loop, test for the NumberOfLEDS = 1:



A: https://makecode.microbit.org/#pub:_KarYJ904xc9a

9. After adding the code that checks for a win, did you end up with two different places in your program that use the same code? Could you improve your code by making the blinking and setting NumberOfLEDs = 12 into a function? What are the advantages of using a function?
   A: The "repeat 4 times" block and setting the pins low/high is repeated in both the forever loop and the "Start Over" (T10) on-touch event blocks. This can be put into a function, so that the resulting code is a little more compact:
   (with functions): https://makecode.microbit.org/#pub:_Xcvi4h87CRVL

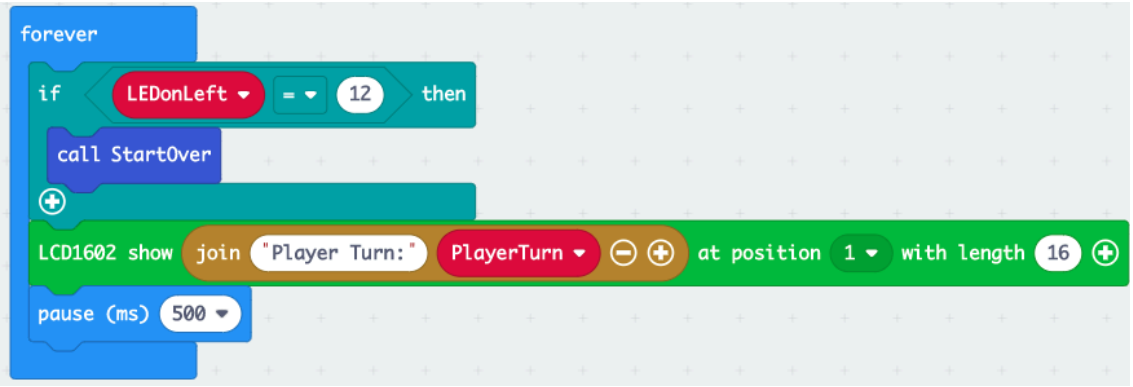
10. Can you program a game that turns off the lights left-to-right, instead of right-to-left? **Hint**: Create a new variable named "LEDonLeft", and when "taking away" LEDs, this time increase the value of LEDonLeft. Here's what one of the on-touch event blocks will look like:



Notice that now you are testing for the LEDonLeft to be less than the last possible position for that move (take away 3), whereas when playing from the right, you were testing for how far you were from “1” the first LED.

Also, when turning off a group of LEDs, because your reference LED is on the left, the LED pin numbers are turned off as the position plus 4, 5 and 6, whereas when you worked down from the right, the LEDs were the position plus 4, 3 and 2.

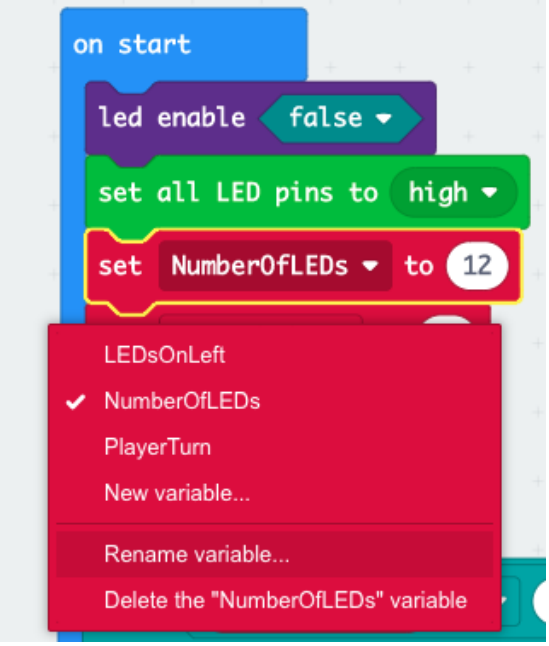
The test for a win will now be to compare to position 12:



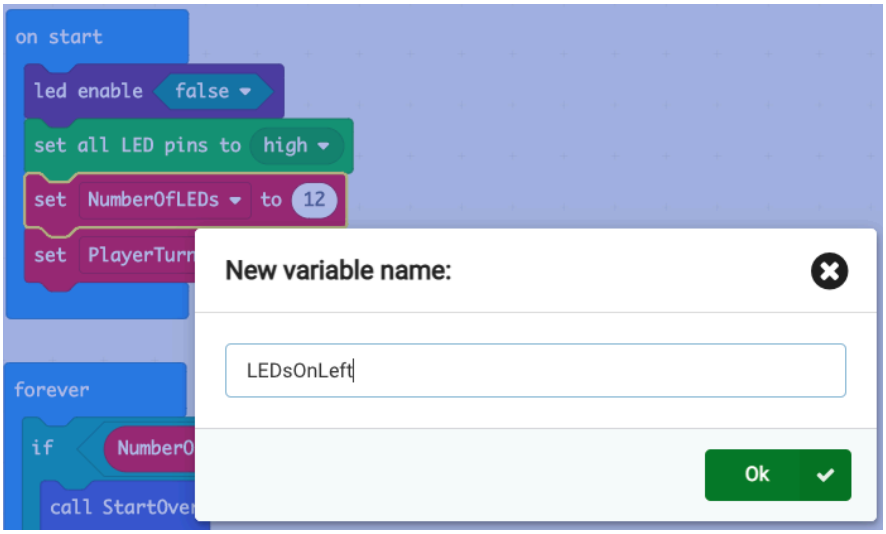
This challenge will not seem like a simple one when you first start, but after you have completed it and look at the "big picture" of the new program, you'll probably then find it overall to have actually been "simple" to change. Learning new skills is often like that. The time early on requires persistence, and may feel like a struggle, but once you've gotten to the end result, you discover that it now feels "easy" and that comfort level will give you new confidence for even greater challenges!

A: https://makecode.microbit.org/#pub:_FvqV0b12J4pg

Programming Tip: When you click on the triangle on a variable, one of the options is to rename it. You can save a lot of time, when you make this program, by editing the variable "NumberOfLEDs" variable in your program from Challenge 9, and changing it to "LEDonLeft". To do that, first click on the triangle for "NumberOfLEDs" in the on start block, and choose, Rename variable…

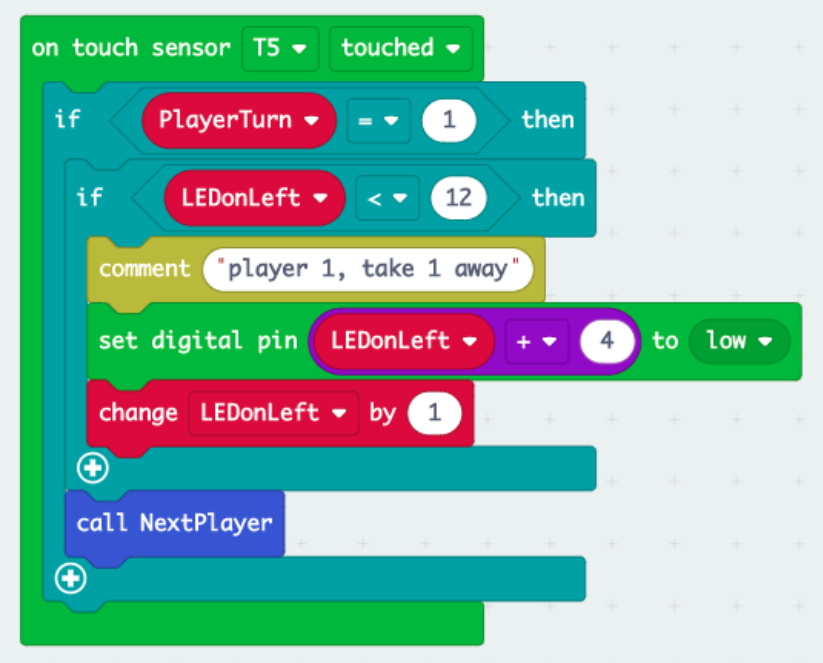


Change the name to LEDonLeft:



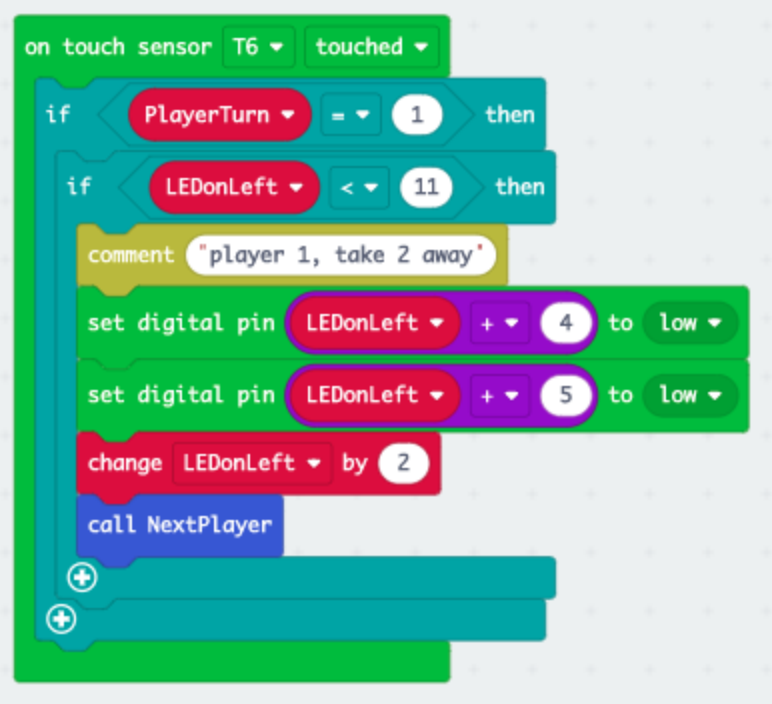
After it is renamed, change the starting value to “1”

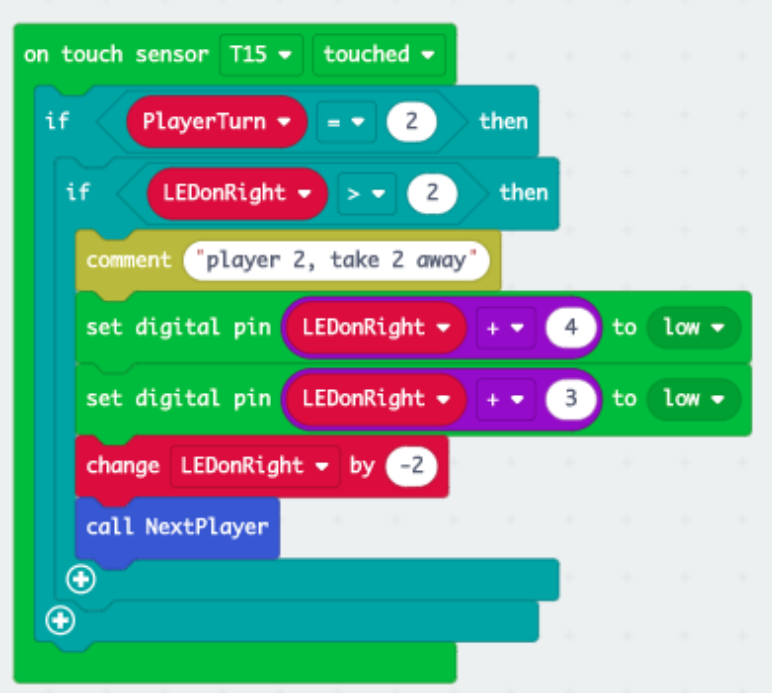
Now notice that everywhere else in your program that used to say “NumberOfLEDs” as a variable name, now show “LEDonLeft”, for example:



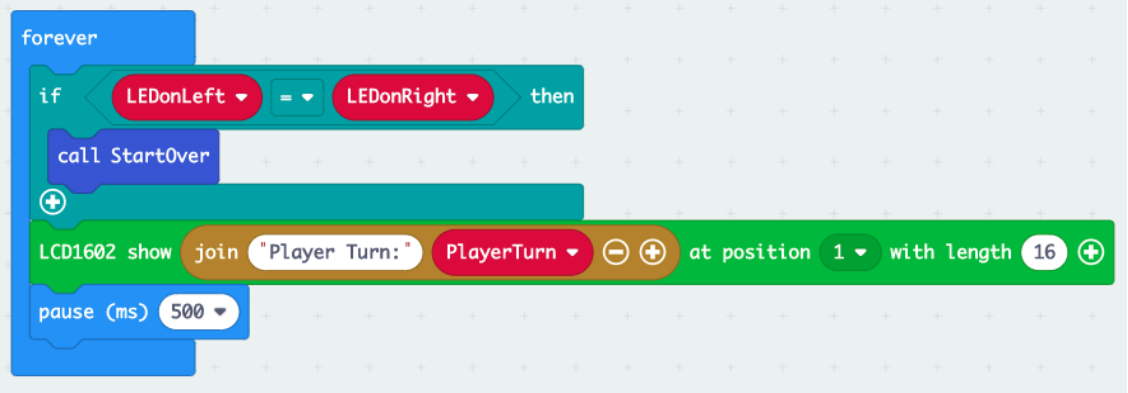
11. Can you modify the program so that the lights turn off, working inward, from the left and right, as each player selects how many to turn off? **Hint:** Create a new variable, LEDonRight, and set it to 12 in the on start blocks:

Then, combine what you did in the "right to left" program with the "left to right" one. This just means the blocks for player 1 will use LEDonLeft, and the blocks for player 2 will use LEDonRight:



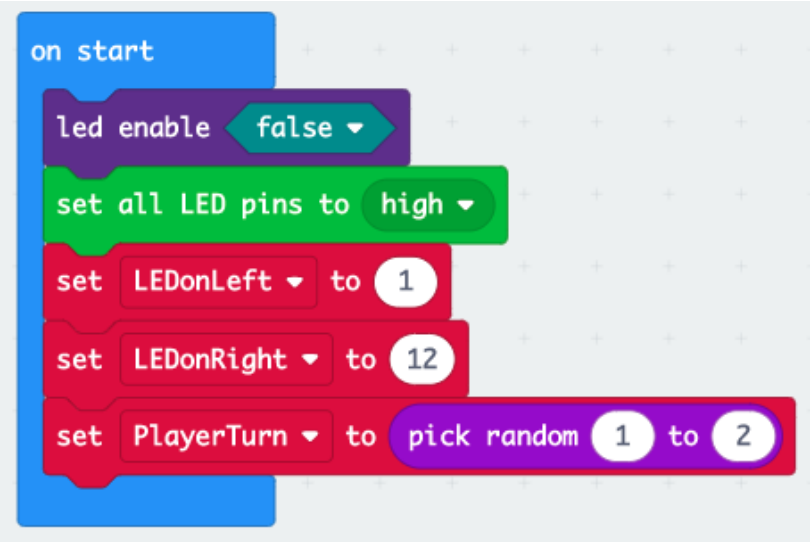


The test for a Win will now be for when both LEDonLeft and LEDonRight are the same, because they're both just the only LED still on:



A: https://makecode.microbit.org/#pub:_TY0CRa59AXwY

12. So that player 1 doesn't always go first, can you use the random number block to "flip a coin" to decide who goes first? **Hint:**



A: https://makecode.microbit.org/#pub:_AFDTm7DvxMTx

13. Could you create a program that would play against a human player?
    A: The number of possible turns and positions is small enough that it could be coded "brute force". So first think about the variables needed:
    a. Player1, Player2 (which one is the human, which the computer)
    b. How many remaining LEDs are still turned on.
    c. Then test for how many LEDs to turn off when it's the computer's turn in response to how many LEDs are still on.
    A: https://makecode.microbit.org/#pub:_dJVcEe35YRLL

d. **NIM strategies**

Remaining LEDs still on...	**Take...**
1 x	(game lost position)
2 x **x**	1
3 x **x x**	2
4 x **x x x**	3
5 x x x x x	(all moves can still lose)
6 x x x x x **x**	1
7 x x x x x **x x**	2
8 x x x x x **x x x**	3
9 x x x x x x x x x	(all moves can still lose)
10 x x x x x x x x x **x**	1
11 x x x x x x x x x **x x**	2
12 x x x x x x x x x **x x**	3

Summary: goal is to leave the **other** player with 1, 5 or 9

14. Games of this sort are explored in a branch of mathematics called game theory. If you have some time, you might want to look into this branch of math some more!
15. Using the technology that you have learned about in this device, what kinds of different other devices or models can you think of that could be made using a new paper overlay?
    A1: "Tug-of-NIM": Use just T5 and T16 to play a "tug-of-war" like game with the LEDs representing the rope.
    https://makecode.microbit.org/#pub:_R88PDwUa3A22
    A2: Use micro:bit radio to have two players, each with their own game box.
    A3: Use the MakerBit network module to have two players, each with their own game box, be able to remotely play from anywhere in the world.

Closing
Time: 15 minutes

Reflection

In a classroom setting, how would you assess the success of this activity? Is there a different assessment for the natural-language "program" compared to the computer program? In looking at the final working computer program, does it indicate anything was in error or lacking in the natural language solution? Does the interactivity of a computer program and the target device provide any advantages to the process that are not possible with natural language alone?

Using the technology that you have learned about in this and previous devices, what kinds of different other devices or models can you think of that could be made using a new paper overlay, and what you now know how to use?
A: This question in association with the NIM game can lead to a good discussion of how a given technology limits some solutions, but where others are possible with some research. An obvious next game after building NIM would be tic-tac-toe, but the problem is tic-tac-toe requires 9 positions of 2 different colors; 18 if done with one-color LEDs. The microbit and MakerBit only have 12 LEDs,, so this would seem to not be possible. However, if a student researches "LEDs" and adds the word "matrix" they may come across "addressable LEDs", which can be assembled in large numbers, and to which the MakerBit can be connected using its I2C pins. Also search for "8x8 matrix" in the MakeCode extensions to see the code support for an 8x8 RGB device that works with the MakerBit.

Finished Coded Programs

Simplest NIM game	https://makecode.microbit.org/#pub:_8a8UwE4rcW6b
Better NIM game	https://makecode.microbit.org/#pub:_D47LuXdFr8VH

Challenges

C1: NIM with some math	https://makecode.microbit.org/#pub:_1EjJ77H6rD02
C2: Block illegal move	https://makecode.microbit.org/#pub:_Vjc7P5arePoK
C3a: NIM game with LCD	https://makecode.microbit.org/#pub:_AyH6LRTEzhmW
C3b: "Center" block	https://makecode.microbit.org/#pub:_iADTHyXb4E0W
C4: Display Player Turn	https://makecode.microbit.org/#pub:_HaL4KMhtVJhy
C5: Players Must Take Turns	https://makecode.microbit.org/#pub:_3tddjKdwVD03

C6: Player Turn Error	https://makecode.microbit.org/#pub:_J7wPEUHvdgaU
C7: Blink Reset (high/low)	https://makecode.microbit.org/#pub:_Dk9PDuX87ix3
Blink Reset (low/high)	https://makecode.microbit.org/#pub:_bdyDwJ9agY59
C8: Test for Win	https://makecode.microbit.org/#pub:_KarYJ904xc9a
C9: Test for Win (function)	https://makecode.microbit.org/#pub:_Xcvi4h87CRVL
C10: Left-to-Right	https://makecode.microbit.org/#pub:_FvqV0b12J4pg
C11: Outside-to-Inside	https://makecode.microbit.org/#pub:_TY0CRa59AXwY
C12: Random Player Start	https://makecode.microbit.org/#pub:_AFDTm7DvxMTx
C13: Computer Player 2	https://makecode.microbit.org/#pub:_dJVcEe35YRLL
C15: Tug-of-War	https://makecode.microbit.org/#pub:_R88PDwUa3A22
Template:	https://bit.ly/2ZuVZJn
2 boards using radio:	https://makecode.microbit.org/#pub:_0pbgihTtqV7K

Standards

California Computer Science Standards		Indiana K-12 Computer Science Standards	New York Computer Science and Digital Fluency Learning Standards	
3-5.CS.1	6-8.AP.13	CSI-1.2	3-5.CT.4	3-5.CT.10
3-5.CS.2	6-8.AP.14	CSII-1.3	6-8.CT.4	6-8.CT.10
3-5.CS.3	6-8.AP.15		9-12.CT.4	9-12.CT.10
3-5.AP.10	6-8.AP.17		3-5.CT.5	3-5.CT.11
3-5.AP.11	6-8.AP.19		6-8.CT.5	6-8.CT.11
3-5.AP.12	6-8.IC.21		9-12.CT.5	3-5.CT.12
3-5.AP.13	9-12.CS.1		6-8.CT.6	6-8.CT.12
3-5.AP.14	9-12.CS.2		3-5.CT.7	9-12.CT.12
3-5.AP.17	9-12.AP.12		6-8.CT.7	3-5.NSD.1
3-5.IC.20	9-12.AP.14		9-12.CT.7	6-8.NSD.1
6-8.CS.3	9-12.AP.16		3-5.CT.9	3-5.NSD.2
6-8.AP.10	9-12.AP.22		6-8.CT.9	3-5.NSD.3
6-8.AP.11	9-12S.AP.22			6-8.NSD.3
6-8.AP.12				9-12.NSD.3