Mission to Mars - USD 383 Summer STEM - Day 3

Summer 2017

Learning Objectives

- Students will explore how a computer represents data in binary
- Students will learn how to convert simple numbers between binary and decimal formats
- Students will see how data encoding can be used to transmit data with very few data points or code words
- Students will see how a data transmission error can cause problems and will discuss ways to fix the problem.

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/day3.html
- Scratch Website: http://scratch.mit.edu
- Binary Numbers on Wikipedia: https://en.wikipedia.org/wiki/Binary_number
- Binary Numbers on Math is Fun: https://www.mathsisfun.com/binary-number-system.html
- Binary Numbers on CS Unplugged: http://csunplugged.org/binary-numbers/
- Binary Flash Cards: http://cse4k12.org/cards/number_cards.html
- Binary Worksheets: http://cse4k12.org/binary/counting_in_binary.html
- Rover Hex Code: https://scratch.mit.edu/projects/112910972/

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Print Binary Flash Cards & Binary Worksheets for students
- Make sure **Rover Hex Code** is available on Scratch website.

Schedule

- 9:00 - Welcome & Icebreaker
- 9:15 - Binary Numbers & Hexadecimal - Flash Cards & Worksheet
- 9:45 - Text, Images, etc. as Binary Data (Data Encoding in ASCII, BMP)
- 9:55 - Break
- 10:00 - Mars Message Decoder
- 11:00 - Wrap-Up Discussion

Lecture Notes

Find someone who:
1: Can draw a square using repeat blocks in Scratch
2: Can make and name a variable in Scratch
3: Can create a list in Scratch
4: Can show the block that starts the program
5: Can show you the blocks to use when computing data
6: Can show you his/her age in binary code
7: Can make a Sprite bounce on the edge.

1. [**Icebreaker**] Take a minute to review what was learned yesterday. Some good questions:
    a. Why do computers want to sort data? Why is that useful?
    b. What are some different ways we learned to sort data yesterday?
    c. What are some examples of real-world things could computers be used to simulate?

2. [**Binary Numbers**] Today we are going to learn about how computers store and manipulate data. Does anyone know how it works? <**discuss - hopefully coming to the idea of binary**>.

<<<These slides are from my CIS 115 lecture. They can be adapted to fit the audience as needed>>>

3. [**Stibitz**] The first person to really use binary in a computer was George Stibitz. In 1937, he completed his “Model K” calculator named for the “Kitchen Table” where he worked on it. It was capable of performing addition on two binary numbers.

4. [**Complex Numerical Calculator**] After receiving a full research grant from Bell labs, he was able to compete his Complex Numerical Calculator in 1940, which was able to perform calculations on complex numbers. It was also unique because it was able to perform those calculations remotely. It was attached to a phone line, and when it was demonstrated at Dartmouth College in New Hampshire he used a teletype machine to send commands to the machine while it was in New York. This was the first example of a machine ever used remotely over a telephone line. (We’ll talk more about this when we get to the history of the Internet.)

5. [**Stibitz Video**] Let’s hear the story from George Stibitz himself

    <play video; start at 1:14, stop at 4:20, right after the “Divide one by zero”

anecdote.>

6. [**Binary - Natural Numbers**] **<Slide has many stops; each bullet is a stop>** So, let's take a look at how the math works for binary numbers. As you may already know, binary numbers are simply a way to represent numbers using the powers of 2. Here is an example of an 8 bit binary number. Can anyone tell me what number this is in base 10?

7. [**Binary - Natural Numbers**] As you can see, for each space that has a one you add that power of 2 to the answer, so, in this case we have a one in the 32 position, the 8 position, and the 2 position.

8. [**Binary - Natural Numbers**] Putting it all together, we have 32 + 8 + 2 = 42.

9. [**Binary Flashcards & Worksheet**]
    a. Gather the students together on the floor or somewhere easy to work. Give each student one set of the binary flash cards from http://cse4k12.org/cards/number_cards.html. This set of activities is useful: http://csunplugged.org/binary-numbers/
    b. Have them lay down the cards in order with the most dots on the left and the least on the right. Then, to make a binary number, add up the dots on the face-up cards to find the value. Face up cards are 1, face down cards are 0. Start with small numbers first, then go up. Usually works best to first start by turning cards up and seeing if they can add the dots, then go the other way by giving them a number and letting them turn over cards to represent it.
    c. Especially focus on numbers less than powers of 2, such as 15, 31, 63. See if they can find the pattern.
    d. On the board, briefly show binary addition. Many of the same arithmetic rules apply to binary.
    e. You can also discuss place values and compare it to the decimal system.
    f. Take a minute to have students fill out the binary counting worksheet, at least to the value of 16. More is better!
    g. Once they have a grasp on binary, introduce hexadecimal. The worksheet makes it easy. Just write "A - F" next to "10 - 15" on the sheet. Explain that it is just another way to refer to the same values, but with only one letter/digit instead of many. There is also a slide to introduce the topic.

<<<More slides from CIS 115 - feel free to adapt or skip parts based on time and audience>>>

10. [**Text**] But what about other data, like words and sentences. For those items, we use a couple of different formats. The first is ASCII or "as-key", which stands for the American Standard Code for Information Interchange. This is a table of all the values in the ASCII code.

11. [**ASCII Example**] Here is a sample of information encoded into ASCII. It is really hard to understand. Let's see if we can decipher it.

    <do an example of deciphering the text a bit>

12. [**ASCII Example**] As you can see, this example says "forty two" in ASCII. There are other formats for this, most notably Unicode, which is used extensively on the internet. It has codes for a much wider variety of symbols and languages.

13. [**Images**] Another common data type that we run into everyday is images. How do you think images are encoded for computers? **<discuss>** There are really two common ways:

    a. Bitmap - each pixel in the image is assigned a numerical value representing the color of the image. This is simple to do, but doesn't work well if you want to magnify the image very far.

    b. Vector - each element of the image is defined via a series of mathematical vectors. This is much more complex (usually must be done by hand), but the images can be magnified indefinitely.

14. [**Vector**] In fact, here is an example of the code within a Vector graphics file. As you can see, it clearly defines in mathematical terms the exact shape that should be drawn to the screen, allowing it to be infinitely scaled in any direction just by applying simple mathematical operations. Most 3D graphics are defined in a very similar manner, but we'll talk about those in a later lecture.

15. [**Bitmap**] Let's look at a bitmap example. This is a 16x16 sprite that could have been used in an early video game. It consists of only four different colors. So, how do you think that would get encoded?

    a. **Discuss! See what they think**

16. [**RGB Colors**] As you may know, bitmap images are comprised of individual pixels, with each pixel giving the color of just a single dot on the picture. To store that data, a computer would simply store the color value for each pixel in an array. The color values are stored using their corresponding red, green and blue values.

17. [**Bitmap Text**] Therefore, this is an example of what a bitmap would look like stored on your computer. It looks like just a bunch of random text, but there is a method to the madness.

18. [**Bitmap Text**] Here is what that text would look like if I actually filled in each area with the corresponding color.

19. [**Bitmap Text with Key**] In many cases, instead of storing the actual color values, the

bitmap might use a color table to store the individual colors, and then use the indexes of the color table to represent the colors in the file. In this case, the file can be stored in less than 4% of the size it would normally take.

20. [**Data Compression**] **<Slide has multiple stops>** The same concept can be used for Text. Here we see a very common piece of text with repeated words. We can easily replace these words with numbers that represent those words instead, resulting in this much short version.

21. [**Mars Rover Hex Code**] Why would this be useful? Well, Mark Watney found out when he was stranded on Mars with no way to communicate with Earth. The only thing that the folks on Earth could do was rotate the camera on top of the old Mars Pathfinder rover. So, knowing that he needed a way to understand, Mark set up 16 signs, one for each hexadecimal character. Since he knew that 2 hexadecimal characters would make one text letter in ASCII, it was a really efficient way to communicate. For our activity today, we're going to build a program that decodes that information just like he did!
    a. Direct the students to load the Rover Hex Code scratch project: https://scratch.mit.edu/projects/112910972
    b. The students should click the "See Inside" button to see the code. If they have a Scratch account, they can also click the "Remix" button to save it to their account.
    c. Let the students observe the program running once to see what it does. It looks like the rover's camera moves randomly, but there is an underlying pattern. Show them how to change the stage backdrop to the circle and observe it again.
    d. Explain that this program is very much a "code and test" programming project, where we try little bits at a time and then run it to see what it does and test it. That helps us avoid little mistakes that become big mistakes later.
    e. Basic steps:
        i. Under the "When I receive Letter" block, first have the program say the direction of the camera. That helps us understand what the camera direction is saying as it rotates.
        ii. Since the camera direction is in degrees, we'll need to convert it into sixteenths. To do that, we divide it by 22.5 (which is 360/16). That will let us know which sixteenth it is pointing to. However, since the direction is negative, we'll get negative sixteenths as well. So, we'll add 8 to the result. We then store that value in the Code variable.
        iii. Next, we need to keep track of 2 consecutive code words. We can do that using the Last variable. So, we'll need a couple of If-Else blocks to check if the Last variable is blank. If it is, we'll put Code into it and get another Code variable from the next movement of the camera. If it isn't blank, we'll use the Code and Last variables to decode the letter and add it to output.
        iv. To decode the letters, refer to the ASCII table here: http://mediawiki.factotumnw.com/mediawiki/images/a/a2/ASCIITable1.jpg. The Last variable is the first hex character, and the Code variable is the

last. All of the characters are in lowercase (and uses tilde for space), so you really only need to deal with Last = 6 and Last = 7. So, you'll have a block of code looking like this:

```
If Last = 6
    If Code = 0
        Set Output to (join Output and ` )
    Else
        If Code = 1
            Set Output to (join Output and a )
    …
Else
    If Last = 7
        If Code = 0
            Set Output to (join Output and p )
    …
Set Last to (blank)
```

v. ***This part takes time***, and students have to be really careful about their code. If they get stuck, look closely at the structure to make sure they didn't get off somewhere. I usually do the "If Last = 6" part first, and show them how to duplicate the If-Else blocks 16 times, then once that is done I show them how to duplicate the whole thing for "If Last = 7" and then they just change the output.

vi. Once it is done, it should give the students the message of "move~eest~mark." Discuss how this message could be very confusing (should it be east or west). How can we fix that? Discuss ways of detecting and correcting errors in the message. If the students press "r" in Scratch, it will replay the message again, this time with the correct direction.

f. See the **MarsHexCode.mp4** video for a demonstration of how to complete this project.

g. Also check the **RoverHexCode.sb2** for a sample solution.

22. [**Reflections**]

a. How do computers represent data?

b. What is the value 42 in binary?

c. What is the value 101101 in decimal?

d. What is hexadecimal?

e. What could happen if a computer message is not correct when it is received? How can we fix that?

- 112910972