Mission to Mars - USD 383 Summer STEM - Day 1

Summer 2017

Learning Objectives

- Students will be able to navigate and use Scratch with a basic knowledge, build scripts, and draw within the program

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/day1.html
- Code.org video: https://www.youtube.com/watch?v=nKIu9yen5nc
- Scratch Website: http://scratch.mit.edu
- Scratch Wiki on Blocks: http://wiki.scratch.mit.edu/wiki/Blocks
- Scratch Spirograph: http://scratch.mit.edu/projects/21326308/

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Have **Spirograph** Scratch file available on the Scratch website

Schedule

- 9:00 - Icebreaker
- 9:15 - STEM Surveys & Accounts
- 9:30 - Videos
- 9:45 - Learning Scratch
- 10:15 - Break
- 10:20 - Shapes in Scratch
- 10:50 - Spirograph
- 11:00 - Wrap-Up

Lecture Notes

1. [**Icebreaker**] Have teachers introduce and give a bit of information about themselves and what they'd like to achieve during the camp. Go around the room and have each student introduce herself/himself and say what they want to learn from the camp. You could also encourage students to ask any questions they have about the instructors or the class, just to set the stage for an open forum of ideas (provided it is PG rated, of course).

2. [**Surveys**] Before we get started, we'd like you to do an online survey about what you know so far. This helps with K-State's research about how to teach students to be computer programmers more effectively.

<<<STEM Surveys and Online Account Setup Here>>>

3. [**Code.org Video**] Computer Science is becoming an important part of many different fields, and knowledge of how computers work could be vital in the future. Code.org is a non-profit organization founded to help bring Computer Science to young people in schools and beyond. Let's take a look at their video promoting what they'd like to accomplish. **<Show Video>**.
    a. Discussion Points:
        i. What do you think about this video?
        ii. Do you think learning how computers work is important? Why/Why not?
        iii. What are some of the things you do every day that use computers? Can you do them without computers?

4. [**The Martian Video**] How can computer science be used in the real world? To help show how useful it is, this week we're going to put ourselves in the shoes of programmers at NASA working on a unique situation: someone is stranded on Mars! How many of you have seen the movie "The Martian"? <get feedback> Let's take a look at this clip from the start of the movie so we know what is going on. **<Show Video>.** This week, we'll tackle several issues presented in the movie and show how computer science can help us bring him home!

5. [**Schedule**] Here's today's Schedule: **<refer to slide>**

6. [**Introduce Scratch**] *Have students load the Scratch website, then instruct them to click on the Create button at the top. That will take them directly to the main Scratch editor. Take some time to describe the different parts of the Scratch editor:*
    a. The Stage
        i. Editing Backdrops
        ii. Editing Sounds
        iii. X & Y Coordinates (can relate back to cartesian coordinates in geometry)
        iv. X ranges from -240 to 240 and Y ranges from -180 to 180 (480 x 360 size)
    b. The Sprites
        i. How to choose a new sprite from the library
        ii. How to create a sprite from an uploaded image
        iii. How to paint a new sprite from scratch
        iv. Duplicating Sprites
        v. Deleting Sprites
        vi. Renaming Sprites (click the blue (i) on the sprite when selected)
        vii. Editing Sprite Costumes
        viii. Editing Sounds
    c. The Palette (http://wiki.scratch.mit.edu/wiki/Blocks)
        i. Motion - Blocks that move sprites around the screen
        ii. Looks - Blocks that change how sprites or backgrounds look

        iii. Sound - Blocks that will play sounds and adjust volume
        iv. Pen - Blocks that will draw on the screen using sprites as the pen
        v. Data - Blocks for storing data into variables and lists
        vi. Events - Blocks for starting and stopping the program
        vii. Control - Blocks for altering the flow of the program
        viii. Sensing - Blocks for learning the state of the program and getting input
        ix. Operators - Blocks for mathematics, boolean, and string operations
        x. More Blocks - Create your own blocks to simplify your programs
    d. Block Shapes (http://wiki.scratch.mit.edu/wiki/Blocks)
        i. "Hat" blocks - they start the program based on the condition specified (ex: When green flag clicked)
        ii. "C" blocks - they have code inside them that runs at specific times based on the block (if, repeat, forever, etc.)
        iii. "Hat" blocks - Stop All, Forever, Delete this Clone
        iv. "Stack" blocks - Standard blocks
        v. "Reporter" blocks - these blocks report values such as numbers, strings, etc. (rounded edges)
        vi. "Boolean" blocks - these blocks are for representing boolean values (angled edges)
    e. Menus
        i. File > Downloading to your computer
        ii. File > Upload from your computer
        iii. Edit > Undelete
    f. Ask for questions / Give students some time (5-10 minutes) to play around and discover how it works on their own.

7. [**Getting Started with Scratch**]
    a. Introduce various blocks (below are some suggestions)
        i. Events - When green flag clicked
        ii. Motion - all
        iii. Looks - say / think, next costume
        iv. Sound - play sound
        v. Pen - clear, pen down, pen up
        vi. Data - (none at this time)
        vii. Control - repeat, if
        viii. Sensing - touching, ask / answer, timer / reset timer, key pressed
        ix. Operators - basic math (+ - * /)
        x. More Blocks - (none at this time)
    b. Work with students to build a simple program
        i. Start by gliding a sprite across the screen with the motion blocks. Have it bounce if it hits an edge
        ii. Have the sprite play a sound when it hits an edge
        iii. Create a second sprite, then have it say something when it touches the

main sprite

iv. Allow students ample time to experiment and learn on their own
v. See **Activity1_Explore.sb2** for an example

c. Students should learn:

i. Simple motion blocks
ii. Working with sprites and costumes
iii. Working within the Scratch environment

8. [**Drawing in Scratch**] See the lesson plans in the "Drawing in Scratch" folder from Nathan Bean: http://www.nathanhbean.com/scratch/ScratchCurriculum/Geometry.zip
    - **a. Students should learn:**
        - i. Using the Pen blocks
        - ii. Using iteration (repetition) to repeat actions
        - iii. Getting user input
        - iv. Simple mathematics in Scratch
    - b. See the **Activity2_*.sb2** files for examples of partially completed projects

9. [**Spirographs**] Now that we can draw in Scratch, let's play around with something even more interesting: spirographs.
    - *a. Show the slides to help explain what a spirograph is. It includes some examples of the math that goes into making a Spirograph work, which ends with the formula for calculating the position of the pencil at any given time.*
    - b. Have the students open the Scratch Spirograph program: http://scratch.mit.edu/projects/21326308/
    - c. Let students experiment by adjusting the value of the two variables. For best results, those values should be "relatively prime" which means that they should not share any common factors. Prime numbers are a good choice to use. Let students share good values that they find with the class.
    - d. **Bonus**: Show students how to modify the variables by hand. Encourage them to try different values outside the normal range (less than 0, greater than 1, etc.)
    - e. **Discussion**: Where would this be important in the real world?
        - i. Artificial intelligence for games - randomly move within a set area
        - ii. Rotational motion - think of the wheels of a steam locomotive
        - iii. Modeling how gears interlock and move together
    - **f. Students should learn:**
        - i. Opening already created projects
        - ii. Working with existing variables
        - iii. Experimenting within a Scratch program

10. [**Reflections**]
    - a. What did we learn today?
    - b. What can we do with this new knowledge?
    - c. What do we want to learn next?

d. Any other questions?

NOTE: I included a bunch of slides on programming. They are good information if you want to show how Scratch relates to real programming, as well as what real programming looks like. The notes for those slides are below:

1. [**Source Code**] Let's say, for example, that we want to write a program that will take an input from the user and print the result of the number divided by 61.

2. [**Scratch**] To do that in Scratch, we would use the following. [slide] As you can see, this program is written in a language that very closely resembles the English language that you and I use every day. That makes it really easy for a human to understand. Do you think a computer is able to easily understand this language?
    a. **Discuss. Ask them why or why not.**

3. [**Language Hierarchy**] In fact, Scratch as a programming language is very hard for a computer to understand directly. Therefore, it is called a High Level language. Let's look at some other examples of High Level languages.

4. [**C/C++**] *I usually note how the main function is like a "hat" block in Scratch*

5. [**Java**] *I usually note how the "class" in Java is like a sprite in Scratch; it can have multiple functions of "hat" blocks*

6. [**C#**]

7. [**Python**] *I used python in a "terminal" style here just to show how it can work without the framework code the other languages require.*

8. [**Other Languages**] There are many other high level languages out there. Some of these you may work with in the future, but many of them you may not need at all unless you are in a specialized field.

9. [**Language Hierarchy**] The next step in the path of most programs is to convert it to Assembly language. This is a language that is still readable by humans, but it is much closer to the language a computer actually understands.

10. [**Compiler**] To go from a high level language to assembly language, we use a program called a "compiler." Its entire purpose is to translate what we wrote in a high level language and turn it into assembly language.

11. [**Assembly Language**] This is an example of what assembly language looks like. This particular part is simply taking a number and dividing it by 61. Looks quite a bit more

complex, doesn't it?

12. [**Assembly Language**] Here's another example from the C program I showed earlier. This is the section of code that is getting input from the user.

13. [**Language Hierarchy**] Once we have the assembly language, the next step is to convert it to machine language. This is the actual "code" that computers can read and use.

14. [**Assembler**] To do that, we use another program, called an assembler, to convert our existing assembly language code into machine language code

15. [**Machine Language**] This is an example of Machine Language code, written in a way that it is somewhat approachable by humans.

16. [**Machine Language**] This is the real machine language code. It is simply a set of binary code (1s and 0s) that tell the computer exactly what to do.