**Scratch Lesson Plans (version 2.0)**

**Terence Soule, University of Idaho,** **tsoule@cs.uidaho.edu**

**Introduction:** These are a series of lessons design for a short, introductory course on programming using the Scratch programming environment. The lessons were originally designed for a one week code camp for middle school students. The participants worked on Scratch roughly 3-6 hours a day. The lessons should take at least 15 hours to work through, but by giving the students more programming time and encouraging more complex programs, they can easily be extended.

The programming lessons build on each other and are designed to be presented mostly consecutively. Some of the other lessons (e.g. storyboarding) can be inserted where it’s convenient. Over the course of the lessons the students create at least two complete scripts. The first is an either an animated scene or an interactive scene (advanced students may combine both ideas. The animated scene has multiple characters that move and interact, and includes dialog, sound effects, changing scenes, and may include visual effects. The interactive scene consists of a scene with objects that the user can click and that respond with movements, sounds, dialog, etc. The second is a simple game based on one of several basic game templates: land a vehicle, catch falling objects, or navigate a maze or ‘minefield’.

**General notes on the lessons:** Each lesson is meant to take approximately 30 minutes of which 10-20 minutes is instruction and the remaining time is for working on the tasks. In addition, students can be encouraged to follow along with the instructor in Scratch during the instruction time. Students can either load and remix the pre-written scripts from portfolio of programs in the *Dign IT Code Camp* studio (described below) or the instructor can walk them through the creation of a new script similar to the one’s in the portfolio. For the simpler scripts it is better to have the student write their own, using the portfolio scripts as a reference when necessary. For the later, more complex scripts, it may take too long to write them from scratch; instead have the students load the script from the portfolio.

**About Scratch:** Scratch (scratch.mit.edu) was developed at MIT Media Labs as a tool for early introduction of programming. It uses a graphical, drag-and-drop programming interface, with easy to understand graphical icons representing basic program elements (see Figure 1). Programs focus on manipulating “sprites”.

Scratch is free and runs as a web application or as a downloadable, standalone application. It has a large on-line community, with a strong emphasis on sharing and “remixing” projects. This on-line community emphasizes programming as a cooperative, social activity, with lots of opportunities to share, discuss, and modify programs. It is also an excellent environment for exploring programming by encouraging the sharing ideas and methods. Programs can be stored indefinitely, and without cost, on the Scratch web site, allowing participants to continue to develop their projects after the course has ended. Programs can be shared via the Scratch website or embedded in regular webpages, making it easy to present projects to family and friends. Scratch has simple interfaces for drawing, and for incorporating sound and images, allowing users to easily create interesting, multi-media programs.

The Scratch programming environment (see Figure 1) is divided into panes. There are separate panes for the current project, the sprites and backgrounds in the current project, and for the scripts or costumes associated with the current sprite or background. Programmatic elements are dragged from a central pane into the scripting pane. Programmatic elements are grouped by type and tabs allow the programmer to switch between groups. Sample groups include Motion, which contains elements such as *turn*, *move*, and *glide*, Looks, which contains elements to change a sprite’s costume and color, and Control, which contains loops and conditionals.

**Scratch Accounts:** To create an account go to scratch.mit.edu and follow the instructions under *join Scratch*. Setting up an account is free. Users must enter their age and an email address. Users under 16 are asked for a guardian’s email address, so in setting up a course it is useful to get parent emails for the students to use. Alternatively they could all enter an email address for the institution hosting the lessons.

**Sample Projects:** A studio of the sample projects used in these lessons is available at <http://scratch.mit.edu/studios/211999/>. Alternatively, all of the sample projects (and samples of student work) can be found by searching for “Dign IT”. The sample projects are labeled *Script N – Name*, where N is the script number and name is the associated name.

**Timer:** it is very helpful to use an electronic timer to keep track of the students’ programming time. The timer should be visible to the students and have an audible alarm. With a timer students can see how much time they have for programming before the next lesson and the audible alarm makes it easier to get everyone’s attention to start the next lesson. A timer widget works well in Windows. After finishing discussing a topic, announce to the students how much time they have for programming before the next topic begins and set the timer.



Command groups

A sprite

Scripts, costumes, sounds tabs

Commands

Project’s sprites and backgrounds

Current scripts panel

Current scene panel

Figure : The Scratch Programming environment. The upper left panel shows the current scene. The lower left panel shows the project sprites and backgrounds. The rightmost panel shows the current scripts. The center panel shows the available commands. The top of the center panel has tabs for scripts, costumes, and sounds. The middle region of the center panel allows the programmer to select between different groups of commands, e.g. commands for Motion, Looks, Sound, etc. The bottom of the middle panel shows the commands available (in the command group that is selected). Commands are dragged from the middle, bottom panel in the current scripts panel on the right.

**Supplies:** Each student or team should have a computer with access to the internet. Optional supplies includes:

A microphone for recording sound clips.

A digital camera for taking pictures to imbed in Scratch scripts.

A drawing program for creating images for Scratch scripts.

Paper and pens or markers for students to draw pictures to embed in scratch scripts. (This also requires a digital camera or scanner to digitize the pictures.)

A drawing tablet.

**Teams:** The lessons are best done with teams of 2-3 students, although students can work alone or in larger groups. For larger groups specific roles should be defined (programmer, tester, designer, etc.) and students should be encouraged to rotate trough the roles so that every student has the opportunity to try every role. Unless there are serious problems, the groups should remain the same between lessons as the lessons are designed to build upon each other leading to the two final projects. Keeping the groups consistent allows the members to continue to work on the same project.

**Student Tasks and Projects:** Each lesson ends with student tasks. These are meant to focus the students during their programming time, not to be assignments that the students turn in. The two student projects, the scene and the game, could be treated as graded assignments in a classroom setting. However, if they are treated as graded assignments it would be beneficial, particularly for younger students, to have some intermediary feedback so the students know if their projects are sufficiently complex and well designed. For the original Dig’N IT code camp the students presented both projects to parents, other students, visitors, etc. on the last day of the camp, but they weren’t graded.

At the end of this document are two design templates that can be given to the students to help them get started on the projects. The first is a blank storyboard that students can use in planning their first project. The second is a blank project form that students can use in planning their game.

**Lesson 1: Introduction to Scratch**

**Topics:** The Scratch website, website accounts, running scripts, *see inside* option, editing scripts.

**Script:** Script 1

**Notes:** Go to the Scratch website (scratch.mit.edu) and have the students set up an account. Younger students will need to enter a parent or guardian’s email. Walk the students through the Scratch site.

Show the students:

* How to find, run, and remix other Scratch projects
* How to search for the Dign IT (or other course) portfolio
* User’s folders
* Sprites: scripts, costumes, and sounds tabs.
* Backgrounds
* Show code menus, drag and drop, play and stop buttons.

**Student Tasks:** Create a team account. Make sure the students use an anonymous name and explain how sharing will work. Run Script 1.

**Lesson 2: First Script**

**Topics:** How to use Scratch

**Script:** Script 2 – Generic Script

**Notes:** Show the students how to create a basic script or have them load Script 2 and walk them through it. Show how to load sprites from the sprite library. Explain how the code groups work and how to drag commands into scripts. Discuss (and demonstrate) the difference between *go* *to* and *glide* in the Motion group. Show how to use the *say* and *think* commands in the Looks group. Demonstrate the difference between *say* and *say for X seconds*.

Code topics:

* Sprites and sprite library
* Start flag and stop sign
* Script menu for Sprites
	+ Events group: *when [flag] clicked*, *when I receive [message]*
	+ Control group: *wait* (for timing)
	+ Motion group: *go to, glide*
	+ Looks group: *show, say, think*
* Movement and coordinates (point out how to use the mouse to find coordinates in the stage, it may also be useful to show the coordinates background)
* Timing
* Backgrounds and background library

**Student Tasks:** Add additional lines (say), movement, and a background to Script 1.

**Lesson 3: Timing versus Broadcasting**

**Topics:** Coordinating sprites

**Script:** Script 2 – Generic Script

**Notes:** Discuss two ways to coordinate sprites. First, by using the *wait* command (in the Control group)) to have the sprites alternatively wait for the other sprite to finish. Second, by using *broadcast* and *receive* commands (in the Events group) to have sprites wait to receive a message before continuing. To help illustrate this idea it’s useful to use a whiteboard to sketch two sprites and their alternating parts in a conversation. Use arrows to show when the ‘action’ switches from one sprite to the other; discuss how waiting or broadcast/receive can be used to govern the switching from one sprite to the other.

 With the broadcast/receive approach each *receive broadcast* statement should be at the head of its own mini-script. This can be used to introduce the idea of parallelism/concurrency – all of the scripts are active, but most of the scripts are waiting to receive the proper broadcast.

Code Topics:

* Wait
* Broadcast/receive

**Student Tasks:** Add more dialog exchange to the scene using waiting or broadcast/receive.

**Lesson 4: Animation and Loops**

**Topics:** Animating sprites, loops

**Script:** Script 3 - Animation

**Notes:** Have the students load Script 3 and walk them through it. Show the different costumes on the costumes menu and point out how they are slight variations of each other. Show the animation loop, which uses the *repeat* loop from the Control group.

Discuss how animation works in general: frames per second and how the brain interprets the separate images as one moving character. Discuss parallel scripts: having an animation script and a separate movement script. Show how the mouth animation worked in Script 1.

Show sprites in the library with multiple costumes for animation – when you select a sprite form the library the number of costumes is shown below the sprite.

If there is time load an animated sprite and walk the students through writing a basic animation script.

Code Topics:

* Loops - repeat
* Costumes

**Student Tasks:** Play with the animation in Script 3. Try different wait durations and movement speeds. If there is extra time they can create their own animation using one of the sprites with multiple costumes.

**Lesson 5: Effects**

**Topics:** Effects

**Script:** Script 2 – Generic Script

**Notes:** Load script 2 and discuss how the final effects are applied using the *change effect X by Y* command in the Looks group. Start a new script and walk through the different effect options. Explain how a loop is used to iterate through the effect values. Use this discussion to reinforce the previous lessons on loops.

This is a good lesson to introduce the idea of a setting up a script and, more generally, a program. Explain the need to set the initial conditions for scripts and programs. In this script the effects have to be reset at the beginning of the script.

Code Topics:

* Effects
* More on loops
* Set-up parts of a program versus body/runtime parts of a program

**Student Tasks:** Add some effects to their on-going script.

**Lesson 6: Drawing sprites**

**Topics: Costumes tab, the drawing programs**

**Script:** New script

**Notes:** Start a new script. Load a sprite from the library, copy it in the costumes tab. Modify the copy to create an animation. Add a loop to show the animation. This doesn’t have to be a great animation – the goal is to show how to draw and/or modify sprites.

Create a new sprite from scratch using the paint program to draw the sprite. Explain the difference between bitmap and vector drawing. Show the different tools that are available. Make sure to point out the zoom buttons, undue buttons, and fill option.

Point out the + symbol marking the middle of a sprite. Explain how, if a sprite is drawn off center, rotating it can also cause it to seem to move – because it rotates around the center of the image (the +) and not the center of the drawn portion of the sprite.

Code topics:

* The painting window
	+ Bitmaps
	+ Vector graphics

**Student Tasks:** Draw a sprite, copy it, modify the copy to create a simple animation.

**Lesson 7: Loading Images**

**Topics:** Loading images, editing images

**Script:** New script

**Notes:** Start a new script. Load a background from an image file (not from the backgrounds library). If there’s time use photo editing software to edit the image (adjust the color, crop it, it) before loading it as a background.

Create a new sprite by loading an image file (not from the sprites library). Once the image for the sprite has been loaded show how erasing part of the image in Scratch actually makes that part of the image transparent. Show how the paint option can be used to further modify the image. A good example is to load a picture with a face, erase everything except the face, then add a stick body. If there’s time, duplicate the sprite and modify the stick body in the copy to create another animation.

Code topics:

* Loading images
* Modifying images
* Erasing and transparency

**Student Tasks:** Incorporate an image into the scene, either as a sprite or a background.

**Lesson 8: Planning a Scene**

**Topics:** Storyboarding

**Script:** None

**Notes:** Because this lesson is not directly about programming it can be presented in a different order. It’s best to include it when the students have seen most of the features of Scratch that they might want to include in their scene (animation, effects, etc.), but early enough that the students haven’t gotten too far along in their scene. This lesson shouldn’t have much instruction time, most of the time should be for the students to plan and work on their scene.

Discuss storyboarding how it uses one quick sketch per major scene. Storyboarding is also a good time to choose the main media forms: pre-programmed sprites and backgrounds, drawn, or photographed. Storyboards should also include notes on media, effects, sounds, etc.

**Student Tasks:** Create a storyboard for their scene.

**Lesson 9: Sound**

**Topics:** Sounds

**Script:** Script 2

**Notes:** Load Script 2 and add different sound effects from the Sound group. Demonstrate the four major sound options in Scratch. First, pre-recorded sounds – go to the sound tab, load sounds from the library, then use the *play sound* command in the Sound group. Second, user recorded in Scratch – go to the sound tab, record a sound, then then use the *play sound* command in the Sound group. Third, loaded from a file – go to the sound tab, load a sound file, then use the *play sound* command in the Sound group. Four, use the sound functions: *play drum*, *play note*, *change tempo*, etc. Discuss the difference between *play sound* and *play sound until done*.

 A possible additional topic is the use of sound for special effects. For example, combining changing size and changing volume can produce a very realistic effect of approaching or retreating sprites.

Code topics:

* Play sound
* Play drum
* Play note
* Change volume
* Change tempo

**Student Tasks:** Add one or more sound effects to their scene.

At this point the lessons switch from creating scenes to creating games. It’s a good idea to give the students a little extra programming time so that they can complete their scene before starting their games.

**Lesson 10:** **Controlling a Sprite**

**Topics:** key pressed, conditionals and sensing, glide to, mouse x,y,

**Script:** Script 4a – Sprite Control Keyboard and Script 4b – Sprite Control Mouse

**Notes:** This lesson is divided into two scripts, which demonstrate using the keyboard and the mouse to control a sprite.

Load and run Script 4a, discuss the loop for keyboard control. Point out the *if* command (from the Control group) and how it requires a condition. Show the *key X pressed?* condition in the Sensing group. This a good place to point out or reiterate that the shapes of the commands matter – e.g. *if* has a rhombus shaped hole to match the *key X pressed?* condition’s shape. As another example of a conditional discuss the use of *wait until (mouse down)* in the script.

Load and run Script 4b; contrast the control loop for Script 4b to the control loop for Script 4a. Show how *glide to* can take variables, e.g. *mouse x* and *mouse y*. Explain how the *glide to* command is used instead of *go to* so the sprite doesn’t simply jump to the mouse. Demonstrate how changing the time in the *glide to* command leads to faster or slower movement.

If the *glide to* command is only based on the *mouse x* (or *mouse y*) value and the other variable is fixed, it creates a pong like one-dimensional movement controlled by the mouse.

 Given sufficient time an interesting project is to create one sprite that follows the mouse and a second sprite (maybe a bird or butterfly) that glides to a location just above the first sprite.

Code topics:

* Sensing group
* Conditionals (if) and conditions (*key X pressed?* and *mouse down?*)
* Mouse position

**Student Tasks:** Try different keys for control, change the step size in the keyboard control, change the time in the *glideto* command, add new movement commands.

**Lesson 11: Touching Colors**

**Topics:** Touching color, variables

**Script:** Script 5 - Touch

**Notes:** This lesson introduces the sensing control touching a particular color. Be careful to point out how the *touching color X* condition is used in conditionals and loops. Note that the color is selected in the code by clicking on the color box in the conditional and then clicking on a color in the script. Because the script uses the background it’s a good idea to show how to add bubbles and other objects to the background using the eyedropper tool to select the correct color.

 This is also a good lesson to introduce variables. Create a *lives* variable (from the data menu) and have it decremented every time a bubble is touched. Be careful to discuss the difference between local and global variables, i.e. visible to only the current sprite (local) or visible to all sprites (global). This is also a good opportunity to reinforce the idea that a program needs to be properly initialized, in this case by making sure lives starts at some reasonable value (3 or 5).

Code topics:

* Sensing group (*touching color X*)
* Variables

**Student Tasks:** Add additional bubbles to the script, then try adding other objects with the correct color.

**Lesson 12:**

**Topics:** variables, gravity, momentum

**Script:** Script 6 – Generic Lander

**Notes:** This lesson introduces a number of concepts. It may be helpful to divide the content into two lessons.

This lesson uses color touching to determine whether the helicopter has landed on the ground (black) or the landing pad (green). Explain the use of the *or* in the conditional in the *wait until* statement in the helicopter sprite.

The use of the *state* variable to track the helicopter’s state (flying, landed safely, or crashed) should be explained as a common coding approach.

This is the first script to include scripts as part of the backdrop. These should be pointed out.

The sprites and backdrop scripts interact via *broadcast receive* pairs. Tracing these interactions on the board may be helpful.

In addition to the new coding commands, much of the lesson should focus on variables, the *X* and *Y* velocity of the helicopter, and the math controlling the helicopter’s movement. Describe how the helicopter’s X and Y movement can be calculated independently. This is a good opportunity to discuss vectors, if the audience is old enough. Also present the equations of motion controlling the helicopter’s movement, particularly how acceleration due to gravity is applied.

 The script has several other features worth pointing out. First, any new backgrounds are automatically included in the game. So, new levels can be added just by creating new backgrounds. Because black is the “crash” color any object outlined in black is automatically an obstacle. Second, the flickering fire effect after a crash is a useful example of animation. Finally, the tipping of the helicopter is useful visual cue to the user that they have changed direction. It’s worth pointing out that these little things (the helicopter tipping, the background noise) can make a program much more immersive and interesting.

Code topics:

* Data group - Variables
* Operators group
* Logic in conditionals (or)
* Vectors
* Equations of motion

**Student Tasks:** Add backgrounds/levels, add other keyboard controls (e.g. a big jump in velocity when the space bar is pressed: “retro-rockets”), change sprite to change themes.

**Lesson 13:** Touching Sprites

**Topics:** Touching sprites, cloning sprites

**Script:** Script 7 – Catch Game

**Notes:** This script introduces a number of concepts and may need to be broken into two short lessons. A key feature of this script is the “master” ball sprite, which remains hidden and is cloned whenever a ball is needed. Explain the interaction between the *create clone of myself* and *when I start as a clone* commands, which is similar to *broadcast* and *receive*.

 Explain random (Operators group) and show how it is used to pick the ball’s X-position and costume.

 Point out the use of *repeat until* with *touching Girl2*. The *or* used in the logic that control’s the ball’s *repeat until* loop is worth discussing. As well as how it is combined with the *if touching Girl2* after the loop that controls the score.

 For the user’s sprite point out how the *ygoal* variable is used to keep the sprite from going to high on the screen.

Code topics:

* Clones (control group)
* Touching a sprite
* Random values

**Student Tasks:** Define another ball type.

**Lesson 14:** Sharing on Scratch (if not done earlier)

**Topics:** Share option in scratch

**Script:** None

**Notes:** Point out the share button. Discuss the *Instructions* and *Notes and Credits* boxes and what to put there. This is a good time to talk a bit about web safety and not, for example, putting any private information, including student’s names, into those boxes.

 Point out the number of stars, likes, views, and remixes counts that are on the bottom of each projects page. Point out that once shared projects HTML address is permanent and can be shared in emails, texts, posted on other webpages, etc. (The actual permanency of the links is up to the people maintaining Scratch.) It may also be useful to point out that even non-Scratch members can see and play with shared projects.

**Student Tasks:** Continue working on their game.

Additional Lessons. At this point the students will have seen all three game templates: avoiding bubbles, catch, and helicopter landing, and will need time to create their own game based on one of the temples. These lessons can be presented between coding blocks while the students work on their games.

**Programming Languages**

**Topics:** Non-graphical programming languages

**Script:** Script 2

**Notes:** Compare how Scratch’s graphical language icons compare to commands in more traditional programming languages. Use Processing or a similar language for contrast.

**Student Tasks:** Continue working on their game.

**HTML**

**Topics:** HTML

**Script:** No script, use a sample webpage. If there is a course or camp webpage it can be used as a sample page.

**Notes:** Brief intro to HTML

Discuss webpages in general – files stored on computers, displayed with a browser.

Show *view source* command in browsers.

Discuss matched commands <> </>

Commands to show: head, body, paragraph, link, image.

**Student Tasks:** Continue working on their game.

**Page** \_\_\_\_\_\_\_ **of** \_\_\_\_\_ **Date:** \_\_\_\_\_\_\_

**Title:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Team:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  |  |
| Notes: | Notes: |
|  |  |
| Notes: | Notes: |

**Page** \_\_\_\_\_\_\_ **of** \_\_\_\_\_ **Date:** \_\_\_\_\_\_\_\_\_

**Title:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Team:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**User controls:**

**Sprite movement:**

**Sprite/sprite interaction:**

**Sprite/background interaction:**

**Sprite edge interaction:**

**Variables:**

**Sounds:**

**Other:**