

# *Scratch Video Tutorial*

**Design Brief for MAS 712 with Mitch Resnick, Spring '05**

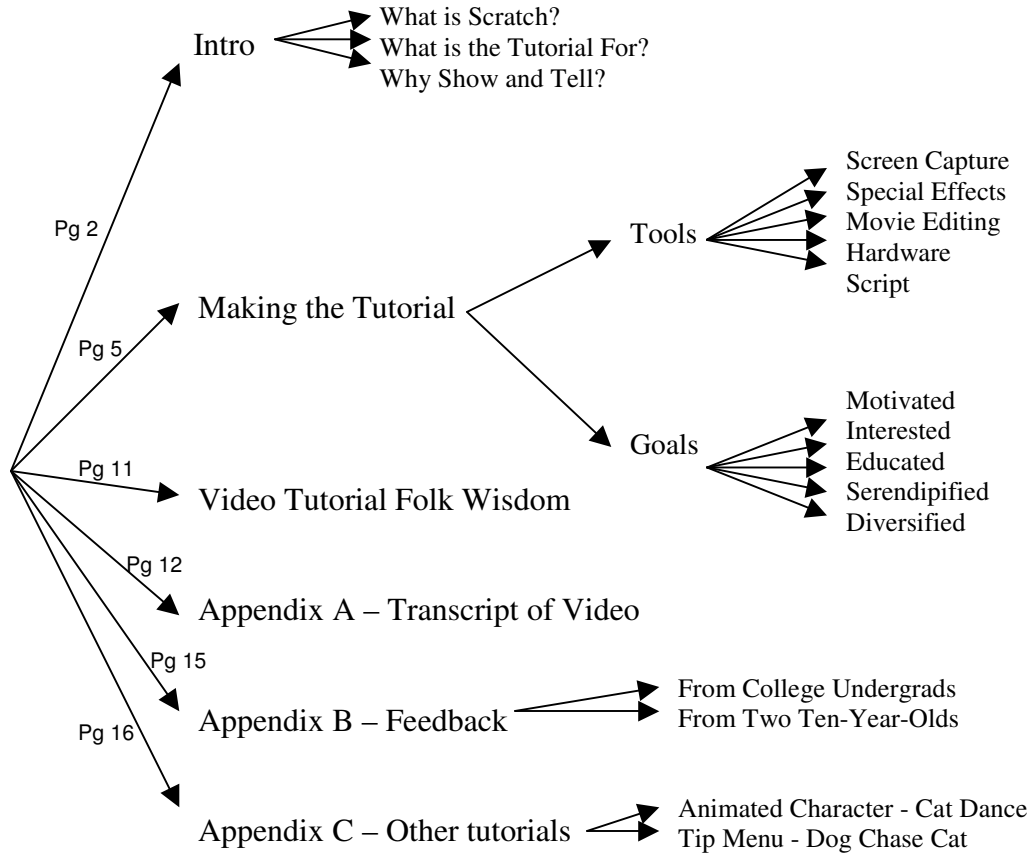
**By Jay Silver**



*I made a video tutorial to catapult people into the world of Scratch. I hope that after watching the tutorial, most kids and their mentors will say something like, "I want to play with Scratch, and I think I know how to start."*

If you're reading this paper, you may also find it helpful to watch the tutorial or play with the Scratch projects. They can all be found at <http://mit.edu/~sil/www/scratch>

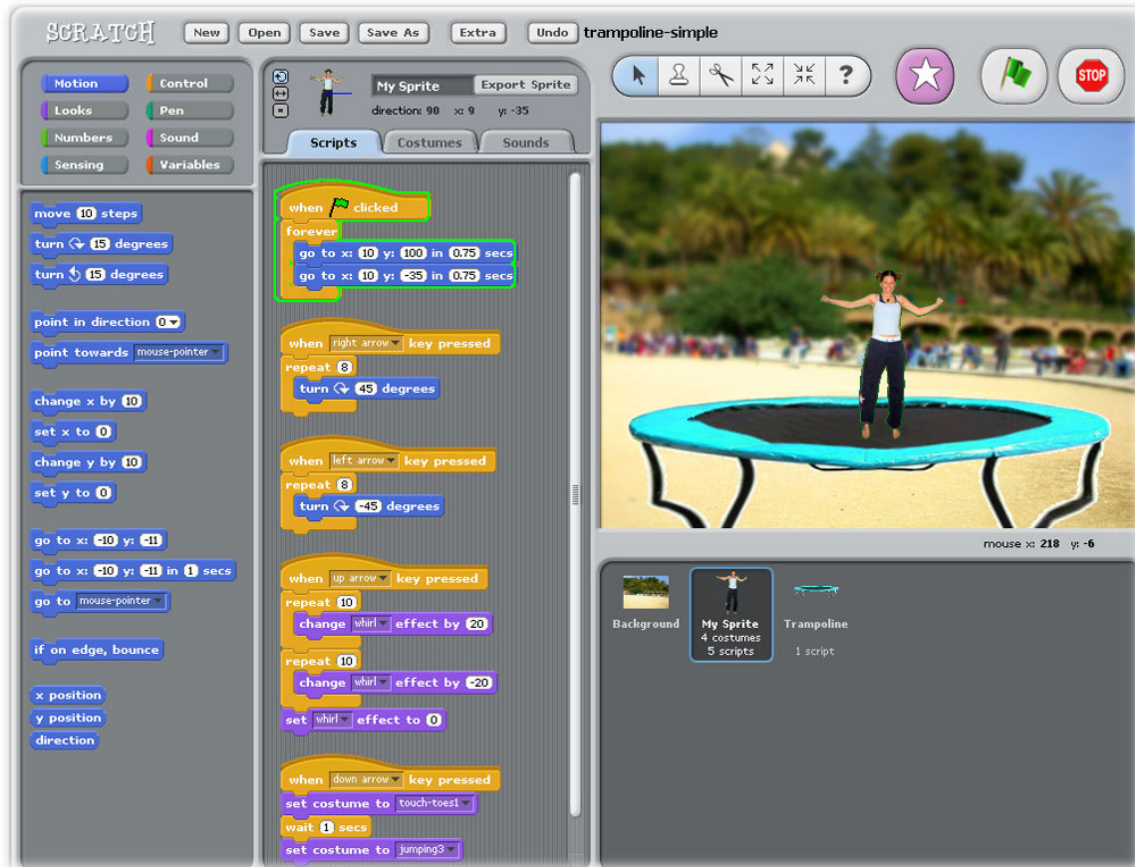
# Document Structure



# Intro

## What is Scratch?

Scratch is a toolkit that lets people create games, animated stories, and interactive art by stacking up virtual blocks that contain computer code instructions. You build Scratch projects on your computer in an environment that looks like this:



You drag code blocks from the left into the programming space in the middle to control what happens on the stage at the right.

A typical project would start with a sprite (character) and a blank background. You would add a costume to your sprite and background, such as the girl costume and the Gaudi Park background costume you see in the example above. Then you might add some code to your sprite, like “When- right-arrow-key-pressed, repeat-8-times: turn-45-degrees,” so that when you push the right arrow key, the girl turns in a circle to do a flip.

# What is the Tutorial For?

The design team was sending Scratch to dozens of Computer Clubhouses across the world, and they wanted the mentors and kids at the clubhouses to have some kind of support to get them started using Scratch. The mentors and kids at these remote locations typically wouldn't have any Scratch experts around, at least not until they themselves became experts. So some sort of documentation would be helpful. In a typical Scratch encounter there is a mentor, a kid, or both, in front of a computer with Scratch. Maybe they have seen Scratch before, or maybe this is their first try. The tutorial is there to help answer questions for the beginners, and hopefully to give new ideas to the more fluent users.

## Why Did I Choose to Illuminate Scratch from the Show and Tell Point of View?

*My tutorial is a video that shows how “I” build a specific project. Choosing the right format was an exercise in perspective.*

I found it helpful to construct a framework for discussing tutorials from the point of view of the learner. Look at the categories below. I'll discuss the categories as they pertain to a Scratch tutorial. The *rules of the game* tutorial would explain what the code blocks in Scratch do as well as any other concepts like costumes, program flow, or sprite layers. This wouldn't limit the type of project the kids might work on at all. But a toll must be paid for the diverse project space: there is no concrete vision of how to proceed. In *rules of the game*, no one is demonstrating how Scratch works, which is why I call it sub-3<sup>rd</sup> person.

- A) Rules of the Game – “Here’s how the different pieces of Scratch work, now try to make something.” There is no example or predetermined project.  
Sub-third-person from the perspective of the learner
- B) Show and Tell – “I’ll show you how I make a project, so you get a feel for Scratch, then you make a project.”  
2<sup>nd</sup> or 3<sup>rd</sup> person from the perspective of the learner.
- C) Step-by-Step – “Here are step by step instructions for how to make a specific project. You try following them, so you get a feel for how Scratch works.”  
1<sup>st</sup> person from the perspective of the learner

To introduce Scratch using *show and tell*, I choose a project, and I show how to build it and how it works when I'm done building it. This is more concrete than *rules of the game*, but at the expense of biasing the learner towards doing a project similar to the one

I show. *Show and tell* is typically a 2<sup>nd</sup> person approach with respect to the learner, however if I (as the maker of the tutorial) tell the learner about someone else's project then it could be considered 3<sup>rd</sup> person.

In the *step-by-step* introduction to Scratch, I would choose a project that I think someone would want to build. Then I write instructions for how to complete each step. You might imagine a variation in which the exact method of how to achieve each step isn't provided, but I would still consider this to be *step-by-step*. The learner does all the demonstrating to herself in this approach, so the perspective is 1<sup>st</sup> person. In a *step-by-step* project, the path is completely determined, which is extremely concrete, but there is no room for creativity. The number of possible project outcomes is 1, unless the learners get rowdy.

So as you move from *rules of the game* to *step-by-step*, the projects become more concrete, but there is less diversity in project outcomes. I chose to use the *show and tell* approach for the majority of the tutorial. You could alternatively call the *show and tell* approach the apprentice approach. I think there's something about apprenticing that seems very socially natural. There's something compelling about the second person demonstration. It lets me take advantage of my own motivations, so that I can honestly show the learner my excitement for my own project. I just can't rally that same exciting point of view if I present Scratch's functions without the context of my personal project. It also means that I can speak in a simple "<I> <action> <object>" syntax, which doesn't have the authoritarian imperative tone of a "<You> <action> <object>" syntax. When I present in the second person relative to the learner, I'm not abstract like a sub-third person demonstration, and I'm not overbearing like a first person demonstration.

Tutorials at all levels of specificity are probably useful for different types of learners and at different points in the learning process. In my case, someone was already developing a help menu that showed how to use each component of Scratch (sub-3<sup>rd</sup> person), and someone else had made a *step-by-step* paper tutorial of how to make a cat chase a ball (1<sup>st</sup> person). There were several example projects available in Scratch (2<sup>nd</sup>/3<sup>rd</sup> person), but no tutorial that showed off a sample project – the projects were just there to be looked at. My tutorial filled this void, and I also prefer this middle ground of learning instruction. I wanted to use a video format, which fits well with a "show how I did my project" style. I tried using video to do a *step-by-step* tutorial also, but due to the screen real estate demanded by Scratch, *step-by-step* isn't a good format for video. Practically speaking, learners may not be likely to switch back and forth between the video and the Scratch project.

I don't necessarily think that every tutorial fits into this 1<sup>st</sup>-2<sup>nd</sup>-3<sup>rd</sup>-sub3<sup>rd</sup> framework, but I found this to be a useful way to organize my approach to the tutorial. I view this tutorial framework as an exercise in perspective taking. I realize that it becomes awkward to call it second person when speaking linguistically in first person, but I think it's worth the extra stretch of my imagination if I can think more like the student.

I use the framework developed here to help analyze some of the other tutorials in Appendix C.

# *Making the Tutorial*

## Tools

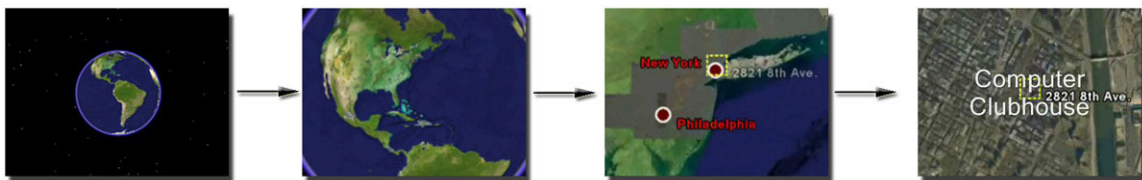
Note: all software was run in Windows XP.

### *Screen Capture*

The majority of the tutorial is made up of video taken from the screen while using Scratch. I used Camtasia Studio to capture video from the screen. Camtasia has a built in option to make mouse clicks audible and visible, which was really helpful for indicating when I had clicked with the mouse. It also allows you to define the exact region of the screen you want to capture. A good screen capture tool is essential to piecing together any software tutorial. Camtasia has a free trial for 30 days.

### *Special Effects*

The beginning of the tutorial has a “zoom in from outer space” effect. That was done using Keyhole. Keyhole is software that allows you to navigate the earth using satellite photos. I set Keyhole to zoom in to a particular computer clubhouse address, and then I used Camtasia to capture the video on the screen. I had to disable hardware acceleration in Camtasia in order to get the video capture to work. Also, I had to fly the desired path in Keyhole several times so that the required images were already cached from the internet. Keyhole has a free trial for 7 days.



There are also some animations at the beginning of the movie. I thought about which software would be best to draw the animations, and I actually decided it would be easiest for me to do them in Scratch, which is sort of ironic. I decided to do a video screen capture of a Scratch project for the animations.



## *Movie Editing*

I did all of the movie editing in Windows Movie Maker 2.0. It's very simple to use, but not the best movie tool available. I had lots of stability problems until I went to tools>options>compatibility and then unchecked all the boxes. I tried to use a pixilated transition effect whenever I came in or out of Scratch. I used a fade transition effect any time that I changed the portion of the screen that the video capture was focusing on. This mimics the feel of a pan and zoom. I used the "double speed" effect to speed up some of the routine Scratch actions, like dragging blocks into the programming space.

I exported the movie file as a .wmv at 768 Kbits/sec and also at 2.1 Mbits/sec. It was important to use at least 768 Kbits/sec in order to keep the resolution at 640 x 480, which was needed in my movie to see the details on the code blocks. That resolution issue is just a quirk of Windows Movie Maker. Windows Movie Maker is "free" if you own windows.

## *Hardware*

I'm going to list the hardware I used to give you an idea of what you might need ahead of time. You won't need the same hardware I used, but I'll give you my suggestions to help you steer clear of any unforeseen bumps in the road:

- I edited the movie on a P4 2.4 GHz with 768 MB. I ran the monitor at 1280x1024. I recommend having a similar or better setup to make the editing process reasonably manageable.
- I used a MuVo Slim MP3 player to record the voice overs. I recommend getting a good quality microphone, since this one had some hissing and accentuated P's and S's. If you record your voice on the same computer you are editing the movie on, then you can use a built in voice over function to make audio-video alignment easier.
- I took the videos with a Canon S230 digital camera (any camera will work)

## *Script* (The entire transcript can be found in Appendix A)

I found that it worked best to write the script as I went along, about 30 seconds at a time. Trying to record the voice overs without writing them down left me concentrating too much on what to say, and not enough on how to say it. I found that it was really

important to keep a lot of energy in my voice to maintain a lively feel in the voice overs. The only way to do this was to write down what I was going to say, and then to actually make the facial expressions and hand gestures I would make as if I were actually talking to a large group of people (which I kind of am, they're just not there with me).

## Goals

I want the tutorial to make people:

- Motivated - To get people to understand why they might want to use Scratch
- Interested - To not have anyone say "I'm bored" during the tutorial
- Educated - To give people a feel for the way Scratch works and what it is like to create a finished project
- Serendipified - To motivate people to want to learn and invent by experimenting
- Diversified - To emphasize many different types of projects

### *Motivated*

The first thing I did was to think of why I get really excited about using Scratch. I decided that one of the biggest motivations for using Scratch personally was that I could take my every day real world and put it in the virtual world. I can take pictures of myself and my friends, put us in Scratch, and make us do things that we can't do in real life, or just whatever I want us to do. Of course you could use Scratch to make up an entire fantasy world too, but the idea of getting myself, my friends, or some famous person I know onto the screen seems to connect with people. So I started out by giving a couple examples of how to do this. First I showed a girl jumping on her bed, and I said, "Say you're hangin out in your room, jumpin on your bed, wishin you had a trampoline." (full transcript in Appendix A)



Then I try to switch the topic over to dancing, but with the same theme of bringing yourself into the virtual world. "Say you're breakin, and you got most of the moves down the way you want them. But there's just that one hand freeze that you can't hold long enough."



I'm hoping that the ideas of trampolines and dancing, along with the idea of bringing yourself into the virtual world, will be enough to motivate many age groups, genders, and cultures to want to try Scratch out. If I've sold them on wanting to try Scratch, then the next thing I'll need to do is keep their interest while I show how Scratch works.

## *Interested*

I made some important tradeoffs between keeping people interested and still covering everything. While you want the scratcher (a.k.a. learner) to come out of the tutorial having learned how to use Scratch, you don't want him to quit watching or stop paying attention from boredom. Interest is perhaps the first and most important domino, because if interest is not achieved then the other goals have no chance of falling in line. In the computer clubhouses, and hopefully in schools of the future, kids aren't forced to work on things they don't find interesting.

The tutorial focuses on a b-girl that can dance back and forth and do some power moves (breakdancing tricks). I try to get costumes on the b-girl, and then get her dancing to a drumbeat as quickly as possible, because then there's something captivating going on – the side to side motion and the music. Someone might argue that the code blocks are the most conceptually crucial part of Scratch, so tutorials should drag blocks onto the screen right away. I can see that point of view.

Getting the costumes on takes 50 seconds, and getting the b-girl dancing takes another 100 seconds. The b-girl dances back and forth using a forever block. The idea is to leverage the forever block to get action going on the screen that will hold people's interest. The idea of "forever" may grab some scratchers' fascination.



One little trick I use to refocus attention is to bring back the “theme music” each time one of the three lessons is starting.

## *Serendipified*

Once the b-girl is moving back and forth, I immediately try to instill a sense of experimentation.

"I love how I can mess with the settings in the stack of code while everything's running. I can change my b-girl's step size. Now she moves 30 steps each time. I can also change her rhythm. She's kinda goin crazy right now, so I'll chill'er out by making each drum last longer."

There's also a scene where I ‘accidentally’ grab the stack of code by the second block instead of the top block. This breaks apart the stack, which is not what is supposed to happen in the tutorial. I say “Woops, gotta grab the stack by the top.” Otherwise I just generally encourage the idea of double-clicking on the block to see what it does. I think I should have done more to encourage experimentation, since I think this is the key to learning how Scratch works. It might have even been worthwhile to state that directly in the tutorial.

I've heard Brian Silverman talk about the first time he saw a turtle do a perfect circle with the command “repeat-36: forward-10, right-10”... or something... and say that he wasn't sure exactly what would happen. Then, just the other day, I saw John working with a girl at the Computer Clubhouse, when she made a butterfly flutter in place in a totally unexpected way using the “if-on-wall-bounce” command. Also, I've personally learned almost everything I know about how Scratch works by trial and error followed by trial and success. If I could change one big theme in my tutorial, it would be to make a bigger effort to impart this nugget of wisdom.

## *Educated*

There are a few basic things about Scratch that have to be learned. I'd say the most important information to impart is the feel of how to use Scratch and what's possible with Scratch, and some of the more advanced Scratchers (people who use Scratch) might pick up on the details. The feel of Scratch is communicated just by watching me drag blocks from the left to the middle each time I need a new block. This is probably the most important concept in Scratch:

Dragging a block in and double clicking it to see what it does.

“What happens if I drag a move block into my programming space and double click on it? The dancer moves 10 tiny little steps to the right. If I get the stamp tool I can copy the move block. I'm gonna change the number in the block to negative 10. Now when I click the copied block, my dancer moves to the left.”

I show where to find the costumes and how to draw your own costume. I also import some sounds. These are some basic functions that are pretty intuitive, but people need to know that they can do them, and maybe the language associated with them. For example, someone might know they want to change how the sprite looks but might not know that the key word is costume. Other vocabulary, like sprite, is introduced in a natural way by using it 5 times right at the beginning. After connecting the word sprite to the other words dancer and b-girl, I drop the term.

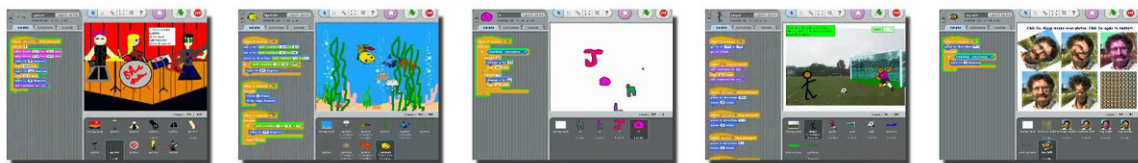
I show a lot of this stuff really fast. In fact, when asked for feedback, three out of nineteen mentors noted that the tutorial went too fast (See Appendix B for raw comments). Still others said it was “very clear” or “very detailed.” I think these kinds of mixed responses exemplify the “you can’t please everyone” property of tutorial making. I think that it’s better to err on the fast side than on the boring side (no one mentioned boredom). I learned this lesson the first time through the design cycle, when I tried to make a *step-by-step* video tutorial. This was so boring, that I couldn’t watch it myself.

### *Diversified*

At the end of the project, I show some possible future directions this project could go in, such as adding more sprites to the mix to create a dance crew.



The important message that I am trying to impart upon departing is that the scratchers should now go and try anything they want. I show some other example projects too.



In the past there’s been a trend toward making video games that keep score. The design team doesn’t feel that this is very representative of the diverse projects that could be done. Also, boys tend to be really into the game projects, which leaves girls on the

sideline. By focusing on self-expression in the tutorial, and by demo'ing lots of possible projects, I hope to lead the Scratchers around the video-game-with-score trap. I think most ages and cultures will find breakdancing to be an interesting topic. But I think the tutorial fell short in the voice overs when it comes to relating to girls. Not that it didn't relate to girls, but that there may be some voice strategies that would relate more.

## *Video Tutorial Folk Wisdom*

By the time I made my last video tutorial, I had a list of guidelines I was keeping in mind:

- Don't pace myself too slow, 'cause boredom is a bigger problem than missing details
- Aim to impart the feel of how to use the program, with the hope that people can teach themselves if they learn how to
- Really get into character for voice overs and put in more effort than you think you need – also, write a script, even if it's only as you go
- Try to get something interesting to look at on the screen quickly
- Use a sound, visual effect, or both, to highlight mouse clicks
- Use music or a special sound at the beginning and end of the video and to indicate new parts of the video
- Think about the needs of both kids and mentors, like how a good cartoon aims at both children and adults

# *Appendix A - Transcript*

Voice over transcript of tutorial follows:

## *Overview and Motivation 0:00*

Hey what's up! Let's take a ride. Down to the computer clubhouse. We're gonna learn about Scratch. Scratch lets you program your computer to do everything you see here and a lot more. You program your computer by stacking blocks on top of each other and inside of each other like this. We'll see how to program later, but first let's see what you can do. Imagine if you could take your every day real world life and put it into a virtual world. Say you're hangin out in your room jumpin on your bed wishin you had a trampoline. What if you could take a couple pictures of yourself, download them to your computer, cut them out, and then take them into the virtual world? You could create that virtual world in Scratch. You might start with a trampoline, then bring yourself into the picture. You can write a program in Scratch to make yourself bounce around and do some tricks, and you could add your favorite background. You can probably already imagine other times you'd want to take your real world into the virtual world. Say you're breakin and you got most of the moves down the way you want them. But there's just that one hand freeze that you can't hold long enough. So you take a picture of yourself, cut it out, and download it to your computer. Then maybe you have another picture of yourself sitting on your computer that doesn't have anything to do with dancing. Take that picture and cut it out too. Then we go into Scratch to make ourselves dance. In Scratch you can hold that hand freeze as long as you want to. You can also make yourself dance in new weird ways. Search the internet to find a cool background and you're ready to go. But how does it all work? How do you make the dancer do exactly what you want? See those stacks of code over on the left? They tell the dancer exactly how to dance to the beat. Now that you've seen a couple of things you can do with Scratch, I'll show you how to use Scratch.

## *Tutorial Part I: Costumes - 2:53*

First I'll open up Scratch.exe. Lemme show how I make a dancer in 3 quick steps. First change the costume. Second make my sprite dance to a phat drum track. Third, give my sprite some power moves. Alright so how do I change the costume? I switch to the costume tab and click import. I can pick my favorite dancer sprite costume. I could even use a picture of myself if I happen to have one. I like this one. If I want to I can paint my own dancer instead of importing a picture. Alright there we go. But I think I like the picture better, so I'll cut the stick figure with these scissors and switch back to the b-girl. If I switch to the background real quick, I can put a graffiti costume in the background. Now I'll switch back to the dancing sprite, and let's get this b-girl dancing. I'll change her name from sprite1 to b-girl.

## *Tutorial Part II – Dance with Computer Code - 4:12*

Now what about the computer code blocks? What happens if I drag a move block into my programming space and double click on it? The dancer moves 10 tiny little steps to the right. If I get the stamp tool I can copy the move block. I'm gonna change the number in the block to negative 10. Now when I click the copied block, my dancer moves to the left. But now I wanna get this girl dancing to a beat. There's a different code under each of these menus. The drums are under the sound menu. The drum blocks snap into place if I drop them right under the move block. The drum block will play whatever drum I choose from the drop down menu. So it looks like we almost have her dancin back and forth with this stack of blocks. Lemme snap all the code together in one big stack. But instead of clicking on the stack every time, what if I could control the code to run forever by itself. I'll put the whole stack of code inside the forever block. Woops, gotta grab the stack by the top. It snaps into the mouth of the forever block if I drop it here. I love how I can mess with the settings in the stack of code while everything's running. I can change my b-girl's step size. Now she moves 30 steps each time. I can also change her rhythm. She's kinda goin crazy right now, so I'll chill'er out by making each drum last longer. There's all kinds of drums we can play... even... Chinese cymbals. Alright let's connect that stack of code to the "go flag" so if someone wants to use my code later, they can just click stop... And go.

## *Tutorial Part III: Interactive Power Moves - 6:43*

Wow we're to my favorite part. I'm gonna give my b-girl some power moves. I'll start with some hip shakin and end with some head freezes and flips. Under the looks menu we can play with some graphics effects kind of like in Photoshop. I can change the color effect. The whirl makes her move her hips. And the fisheye blows her up like a balloon. Lemme reset all that real quick, and I'll throw these blocks back where they came from, and try to get her hips shakin by settin the whirl effect. Now I'm gonna grab some wait blocks, so I can tell the dancer to move her hips one way... wait... and then move her hips the other way. Okay I've got her shakin her hips. If I get one of these blocks I can hook the dancer up so she's controlled by the computer keyboard. Now if someone walks up and pushes the left arrow key, she shakes her hips. I wanna add some sound effects to the hip shakin. Here's a couple sound effect that are already loaded in. I'm gonna add a couple of my own. A bass and snare drum from a human beatbox and two samples from a DJ record I have. I'll get a sound block and set it to play a sample from the DJ record. Then if I break the stack apart, I can squeeze my sound sample in the middle of the code. Check this out, it's really cool. I can copy that whole stack of code just by right clicking. Then I can change the setting on my copy so I get a new power move. The b-girl should fisheye and play a bass drum when I push the "c" button on the keyboard. Now for the last powermove. I'm gonna teach my b-girl to do a head freeze. I'll grab a turn block and see what it does when I click on it. I want her to turn faster than that, so I'll change the degrees from 15 to 60. Now I wanna control my code to repeat the turn 3 times. That way I can get the b-girl to stand on her head. Now I'm gonna hook up the keyboard, so that she does the headfreeze whenever someone pushes the right arrow key. If I click on the background Scratch will clean up my code for me. Now just add a quick sound effect to

the head freeze. Let's see what it all does. Alright! Looks like we've got our final product.

### *Conclusion – 9:56*

Now you can try to build this project yourself and add extra power moves or more dancing sprites. You could also check out the sample projects, there's a lot of them. Or you can make any project you want. It's totally up to you. Thanks for ridin with me. I'll check ya later.

### *End – 10:23*

# *Appendix B - Feedback*

## *Feedback from non-technical College Undergrads*

The tutorial was shown in a class of college undergrads at UCLA who have been working with Scratch and with children. Here are the comments that were collected:

- Good Sample Project.
- Should show a little more of the other sample projects.
- Video is too fast.
- Funny!
- It seemed to tap into something that the kids might want to do--dancing.
- Make things a little more detailed--like cutting the Sprite.
- Interesting and funny.
- Very detailed; narrative simplified Scratch, and gave good examples.
- Didn't really focus on drawing for themselves; focused more on "importing" images.
- Step-by-step instructions of Scratch--definite strength.
- Could use more Scratch examples and less of an intro.
- Very clear and concise.
- Maybe explains a little fast.
- Maybe could use voice over from Clubhouse members.
- Too fast.
- Explain why you would put commands before others or after.
- I liked the sample project you gave.
- Showed us different things that can be done.
- I liked the b-girls, hip hop theme--it grabs attention and is culturally relevant--it draws young audience in.

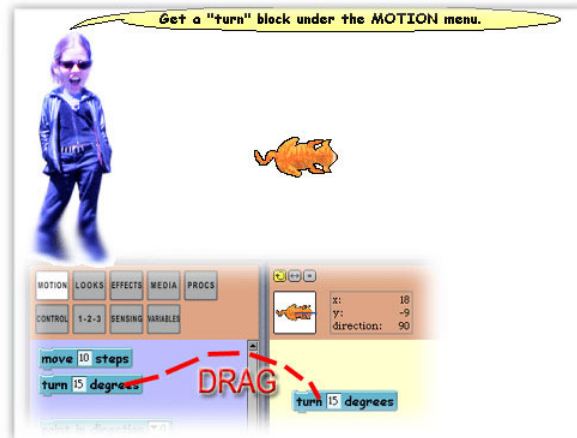
## *Feedback from two 10-year-olds at a Computer Clubhouse:*

I showed the tutorial to two kids at a Computer Clubhouse one day. When John asked them if they thought it was boring, they said no. When he asked if they thought it would be good to help someone learn how to use Scratch they said yes. I tried a slightly open ended question - I asked them what they want to do next. They said, "Try Scratch out." I liked the answer, but unfortunately that was the end of the feedback.

## Appendix C – Other Tutorials

### Step-by-Step In-Scratch Tutorial

I first tried making a tutorial within Scratch. I made a character that talks, dances, and gives instructions with a combination of screen images, voice, and text.



This tutorial fits under the *step-by-step* category. It helps you get the cat to dance back and forth with a drumbeat. The fact that it's within Scratch makes it much more viable as a step-by-step tutorial, because there is no switching back and forth from video... or even from paper. It's all right there on the screen. The tutorial is set up so that each time you click on the girl she gives you your next instruction, which is simple, but it would be nice to be able to go back to previous instructions. The visual cues are embedded in the background image, which keeps them from interfering with the action on the screen. I really like the voice used in this tutorial. It's recorded by my girlfriend who is a kindergarten teacher, and she talks as though she's addressing a classroom of younger children. I think the non-threatening ring to the voice really makes the authoritarian "you do this now" style of the tutorial more bearable. If you are going to attempt a *step-by-step* tutorial I do recommend experimenting with an in-scratch design and a very friendly voice, although paper or html also work.

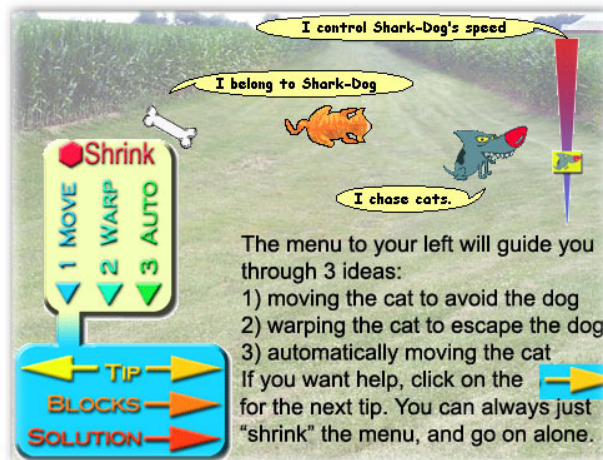
### A System Puzzle Scratch Tutorial

This tutorial was sort of still in the dance world, if you consider a dog chasing a cat to be a dog-and-cat-dance. The tutorial tries to get the scratcher to follow three general steps:

- 1) get the cat to *move* when you push the keyboard keys, then try running away from the dog using the keyboard

- 2) get the cat to *warp* across the screen by disappearing and reappearing somewhere else, and get the cat to stun the dog for a moment
- 3) get the cat to *automatically* run away from the dog while you sit back and watch

This tutorial was also within Scratch, but rather than using an animated character, the help is more passive. There's a menu system that guides the user through the three phases, but the menu can be minimized at any time to allow uninterrupted exploration.



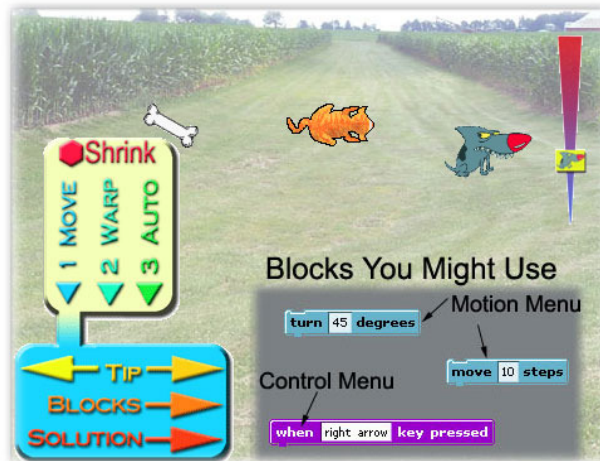
In each step of the puzzle, the menu offers three levels of assistance: tips, blocks, and solution.

## *Tips*

The first level of help is tips. The tips provide instructions and clues if the user wants to try to achieve the objectives laid out. Success isn't defined by a particular arrangement of blocks. But the goal named on the menu, such as *move*, no matter how it's achieved, is really the target. If you're stuck after looking at the tips, or if you just want to see which blocks to use right away, you can click on blocks.

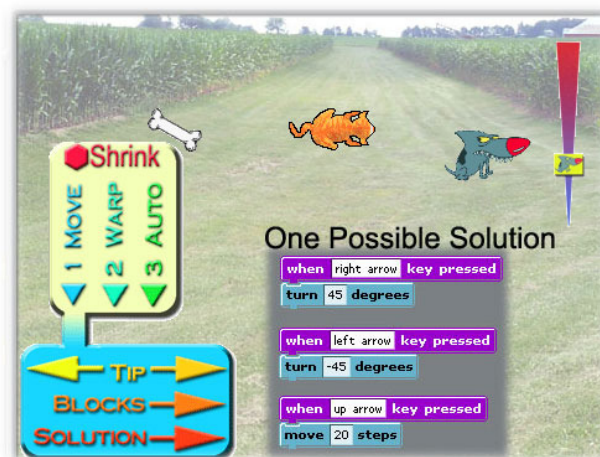
## *Blocks*

The blocks option displays some blocks that could be used in one possible solution. So to make the cat move, one way is to use the turn, move, and "when <keyboard button> key pressed" blocks. The caption says "blocks you might use" because there are many ways to get the cat to move.



### Solution

Each of the three steps also includes at least one possible complete solution. To get the cat to move around the screen, the configuration of blocks below will work.



The scratcher is free to look at the solution right away or to only look when stuck. Also, since there are many possible solutions, the scratcher could look at the solution after creating her own way of reaching the goal.

### Discussion

This is a second way to do an in-Scratch-tutorial. The goals might be interesting to children, but adults and systems educators may also take interest, since the end of the project can result in a system that's in dynamic equilibrium. The project can also have unpredictable results, since it allows people to explore different solutions.

Unfortunately the tutorial itself may also require a tutorial. That is to say, it's not 100% clear how to use the menu. It may also just be generally confusing to have a tutorial embedded within Scratch, since the lines between the "software" and the sprites themselves become blurred. I think the tutorial is a good example of Scratch being used in ways that it wasn't necessarily intended, which speaks to the higher-than-some-people-think ceiling Scratch has. The tutorial uses aspects of first and third person (relative to the learner) in the sense that it tells you to work on a specific project, but also shows you how "I" would do it. The sense of "I," however, is diminished since it is a menu that's presenting the information, so it becomes 3<sup>rd</sup> person.