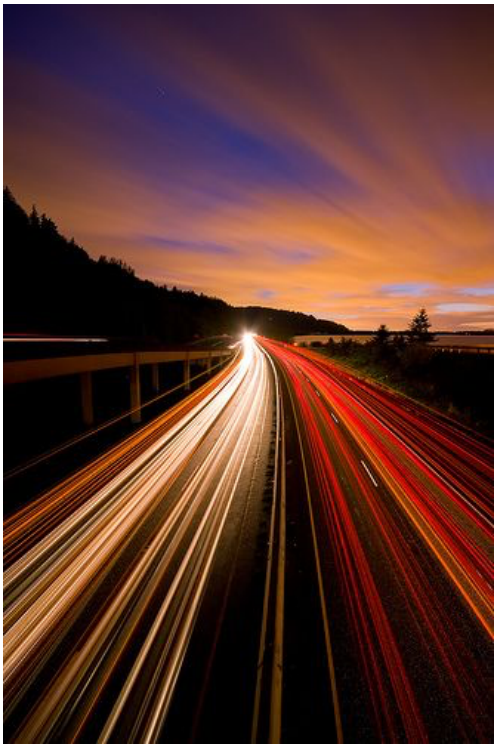
Sphero and Swift Playgrounds

The Sphero-art project: Near the end of our section on algorithms, our middle- and high-school students did a fun project using Swift Playgrounds to control a Sphero (http://www.sphero.com) SPRK+ robot ball. These cute toys connect via bluetooth to iPads and iPhones and respond to commands from the iOS devices. Students can program the robot to roll and flash colored lights in response to the operators commands. Sphero provides demonstration apps that allow users to manually drive their robots and play games. But our computer science students were really inspired when we assigned a project to write an elegant Swift program that would command the Sphero to create a work of art.



Long-exposure photography: We first explored the idea that cameras are light gathering devices, and by holding open their shutters for long durations, they capture the light for an extended period of time. This technique is used to create many artistic representations of light such as cars traveling down a highway and extended exposure thunderstorm photos.

The art: Our students used an iOS app called Slow Shutter ($1.99 on the App Store) - (https://itunes.apple.com/us/app/slow-shutter-cam/id357404131?mt=8) to record the Sphero executing the student's algorithm. Students had up to three minutes to execute and record their custom Sphero algorithm and create their photographic artwork.

The scoring rubric combined both artistic and algorithmic evaluations. Artwork was assessed by our school's art teachers, and elegant Swift code (loops, conditionals, functions) earned extra points. Throughout the project, students worked under this dual-constraint programming assignment, developing and efficiently programming an algorithm that also created an aesthetically pleasing artwork. An additional constraint imposed on the students was that we all

shared one Sphero robot! Students had limited attempts to test their algorithms and had to be particularly careful to mentally trace through their Swift programs and debug them in order to best utilize their limited testing opportunities.

Starter demo code: To initiate this project, we downloaded the demo playground book (https://developer.apple.com/swift/blog/downloads/Sphero.playgroundbook.zip) from Apple's 2016 World Wide Developer Conference that introduced the Sphero/Playgrounds toolset for education. One program in this book directs the Sphero to draw a square, changing the light to random colors at each turn.

```
import UIKit
import PlaygroundSupport

guard let sphero = Sphero.nearest() else {
    print("Unable to find a Sphero device")
    PlaygroundPage.current.finishExecution()
}

let colors = [🟦, 🟦, 🟥, 🟥, 🟧, 🟩]

var i = 0
func rest() {
    sleep(1)
    let color = colors[i % colors.count]
    sphero.setColor(color)

    i += 1
}

while true {
    sphero.roll(speed: UInt8.max / 4, heading: 0)
    rest()
    sphero.roll(speed: UInt8.max / 4, heading: 90)
    rest()
    sphero.roll(speed: UInt8.max / 4, heading: 180)
    rest()
    sphero.roll(speed: UInt8.max / 4, heading: 270)
    rest()
}
```

We ran the Swift code, watched the Sphero trace the square (which changed color at each turn), and then dissected the code.

The students determined that the sphero.roll() function inside the while loop and the rest() function combined to determine the direction, speed, and duration the Sphero traveled, while the sphero.setColor() function determined the color of the Sphero's lights. This was enough to get students started with their own custom artistic creations.

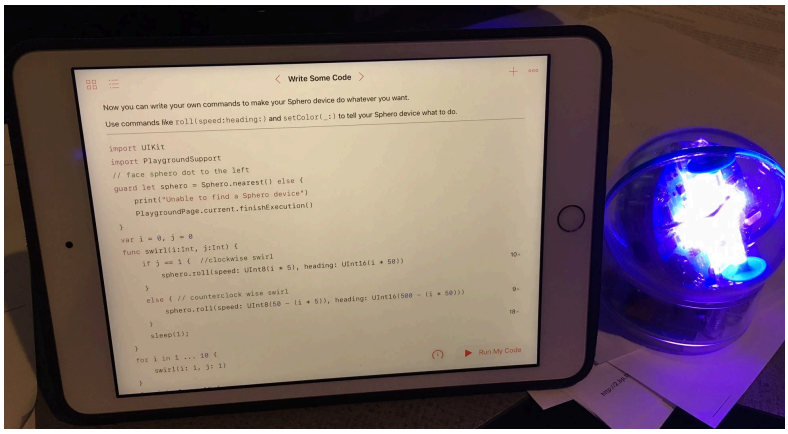
Stepwise refinement on projects: Since we had only one Sphero for the entire class, students worked on their algorithmic designs *'blind.'* Far from being frustrating, these logistical constraints added a level of care to their home coding sessions. In addition, there was additional communication on the class forum where students asked teachers and their peers to 'check-over' their algorithms.

After testing their artistic algorithms, the students found that they needed to better understand the Swift functions that interacted with the Sphero device. They pared down the square program to only one sphero.roll() function and tested many different values for each parameter until they had solidified the units and range limits for each parameter. They answered questions such as:

- What is the maximum and minimum values for the speed parameter?
- How do we control the direction of movement?
- What units does the direction and duration parameters use?
- Does the Sphero continue to roll due to its inertia after the function call, or does it stop immediately?

*A good lesson at this point would be to remind students that sometimes coding can be a high-anxiety activity. NASA programmers, for example, have developed strategies for peer code reviews and testing to minimize risks that their programs will fail with high cost to human and material resources. Their programmers often only have "**one shot**" to get it right. In most cases, their care and attention to detail is rewarded with successful missions advancing the human understanding of our Universe, but in other notable exceptions, even this added scrutiny is not enough to prevent disaster resulting from erroneous software programs.*

- What roll does friction play in the starting and stopping of the robot and how does different surface friction affect Sphero motion?
- Are there limitations to the type of Sphero art that can be created?



Stepwise refinement on algorithms: Our students iterated on their designs by 1) editing their code, 2) mentally tracing their code, 3) physically illustrating their algorithmic designs using paper and colored pencils (at least how they perceived their Swift code to execute), 4) debugging syntax and logical errors, 5) testing their Swift algorithm on the Sphero device, 6) video recording/photographing the Sphero executing their custom algorithm 6) assessing the end result and determining next steps: (anything from going back to the drawing board, to minor edits and tweaks, to freezing their code and derived artwork as final and moving on to help other students).

More than any other lesson in our semester course, this project motivated students to totally abandon their ideas at various points and rethink both their artistic- and software-designs from scratch—a valuable lesson.

Perhaps counterintuitively, the more readily a software developer can discard a non-productive approach to a problem and start from scratch, the more effective they may end up being in the long run.

Final thoughts: The students were proud of their beautiful works of art created by using their Swift programming skills to control the Sphero. The combination of the Swift programming language, the mobile software development environment of Swift Playgrounds on the iPad that permitted students to move around the floor while their software executed, and the colorful and engaging Sphero robot proved a powerful and compelling environment for our students to create custom works of art.

While we did not provide awards to our students at our end-of-project demonstration and celebration, if budget and time permits, you may consider printing your student's algorithmic artwork on a high-quality color printer for them to share with family at home. Or even better, use fractureme,com to make high-quality prints on glass suitable for home display.

In either case, don't forget to have the artists sign their work (using an Apple Pencil) before printing!