Sphero Dance Hero

Prepared by:
Gonzalo Gomez, Israel Gomez, David Hunter, Mike Kuzminsky
Hunter Lang, Ryan McManus, Timm Nygren

CSM CS Field Session 2013

Client:
Wes Felteau

Director of Developer Tools
Orbotix

May 13, 2013 - June 21, 2013

Introduction

Client Description
Our team was tasked with creating an app for the Sphero, a robotic sphere created by Orbotix. This robot can be controlled either by using a free control interface provided by Orbotix, or picked up and used as a controller, on many different mobile devices. Orbotix provides several games, both free and paid, to promote their device, and we were asked to create a similar app. After discussing with our contact, Wes Felteau, we picked a project that resembled games such as Guitar Hero and Dance Dance Revolution, and requires a Sphero connected to that mobile device to play. The idea was to provide a way for users to connect to a central server, choose a song to play, choose a difficulty, and then require the user to move the Sphero in the indicated direction.

Play would be dictated by a set of moves on screen, and the user would be scored based on the number of correct moves hit, as well as keeping track of the number hit in a row correctly. When a player hits more than a certain number of correct moves in a row, a multiplier is increased, and more points are accrued. Scoring is saved, and is displayed upon completing a song. High scores are recorded, and can be saved using a Parse login, or by logging into Facebook.

Product Vision
The product is aimed at those who own Sphero robots, and also will attempt to promote increased sales of the devices by providing a stable, fun gaming app that runs on many different devices, including many iOS and Android devices. The app aims to use Parse integration to provide a cloud backbone for all devices, allowing us to store songs and moves, and also storing to the device when necessary. This integration, because of its cross-platform ability, is much cheaper than needing to support in-house servers, and requires less maintenance in the future.

Requirements

Functional Requirements
The game should provide a unique user gaming experience, and allow social media integration. The app should also:

- Allow the user to create and share songs with friends through Facebook
  - Integration of these challenges from friends either through Facebook or Game Center (iOS only)
- Keep score through levels by storing information on Parse
  - Have the ability to share scores and challenge through Facebook
- Different movements incorporated
  - 4 base movements -> Left, Right, Up, Down
- Gameplay will be based around a central point on the screen, with segmented boxes moving from the outside in
  - Direction from which boxes move is the direction Sphero needs to move
  - Changed from original design of circles moving around a central point (see figure 1 below)
- Introduction level that lets new users experience the UI with approximately 1 minute of our first testing song
- Have variable change that accounts for scores based on difficulty
  - Create an algorithm that accounts for the differences in radii between the target ring and the current directional ring
  - Difference accounts for points based on current difficulty
  - Variable window closes as play gets harder

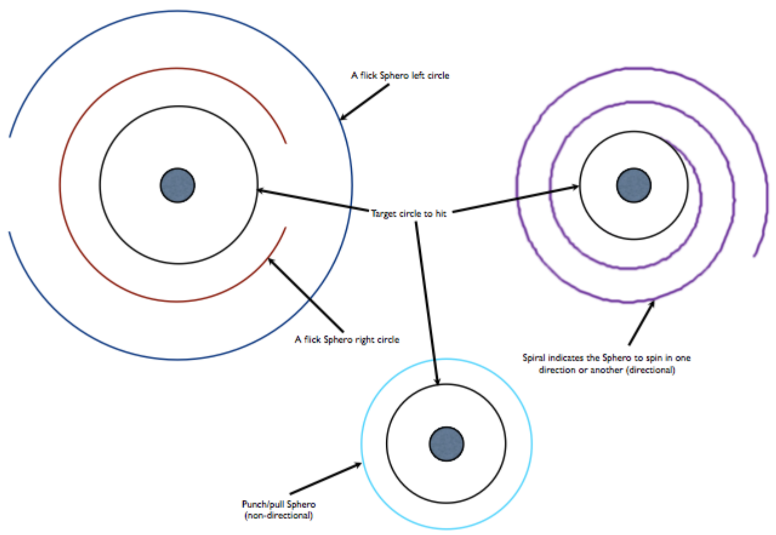
A flick Sphero left circle
Target circle to hit
A flick Sphero right circle
Spiral indicates the Sphero to spin in one direction or another (directional)
Punch/pull Sphero (non-directional)

Figure 1: Original screen layout incorporating multiple move types

- Vibration in Sphero plus (possible) color change to indicate failure to hit note(s)

Non-functional Requirements

- Development cross-platform on iOS and Android using SDKs provided by Orbotix
- Using Parse API for cloud access and Facebook integration
- iOS development in Xcode
- Android development in Eclipse
- Use a NoSQL database (like Parse) to process requests
- Have a simple UI that does not confuse the user
- Storing Sphero timestamp data for timing
- Creation of “sliding window” to evaluate packets across a set time

Risks

- Bluetooth distance failures inherent to the device
- Bluetooth latency issues
- Severe Sphero device failure
- Severe mobile device failure
- App crashes
- Android screen resize failure or non-implemented screen sizes

System Architecture

In our initial ideas for the app, we had designed a finite state machine very similar to Figure 2 below. The idea we had was a working app that would be professional and playable, with a play section, a create section, and the ability to save and load user information, scores, and songs with moves.

Upon opening the app, the user is presented with several options on the main screen.

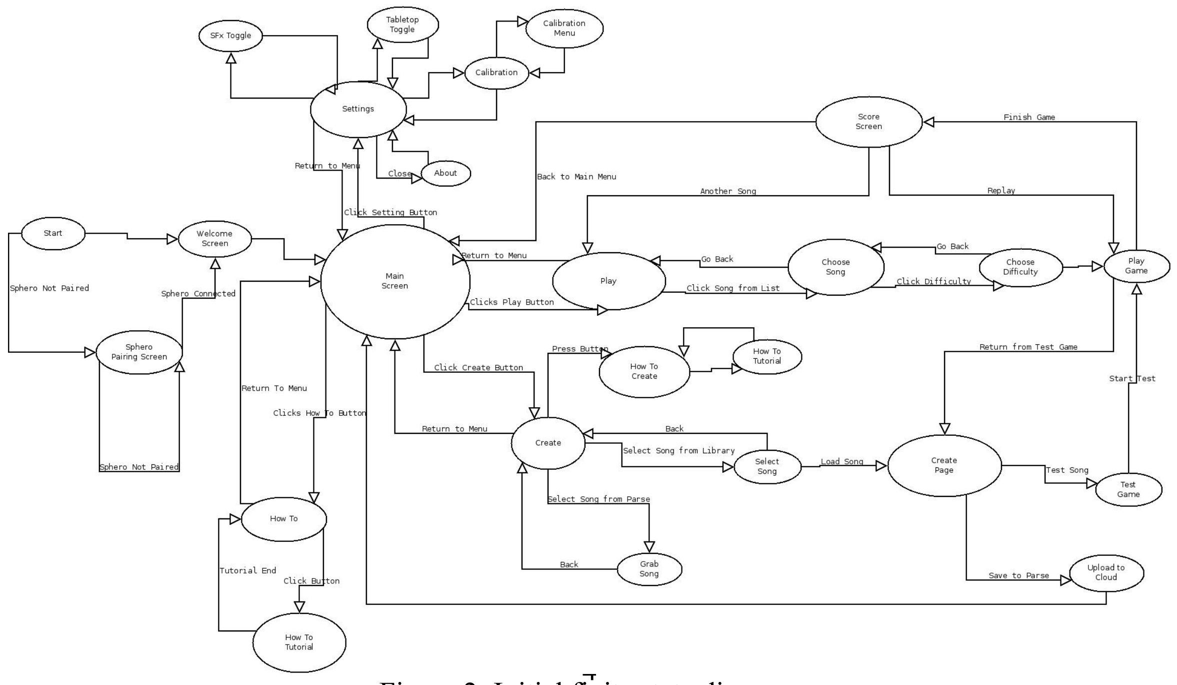

Figure 2: Initial finite state diagram

They can choose to play, learn how to play, change the settings, or create a song. On choosing play, the user can log in using either a Parse user name, or log in using Facebook. They can also choose to play as a guest; however no scores will be saved online. Once a log in method is chosen, the user is given a choice of songs from Parse, along with each associated difficulty. When a difficulty is chosen, the app is then moved along to the play screen, where the user is prompted, with the blue LED positioned under the thumb, to move the Sphero in the directions indicated. Once the song is finished, the user is then pushed to the score screen, where their score and any associated high scores are displayed. The player then has the ability to go back to the main menu, or replay the song.

When we decided to implement the Parse API into our code, it was a decision that was rather easy to make. Using the provided guidelines, with excellent online documentation, Parse integration was simple to use, and allowed us to store songs, scores, and moves online, as seen in Figure 3.

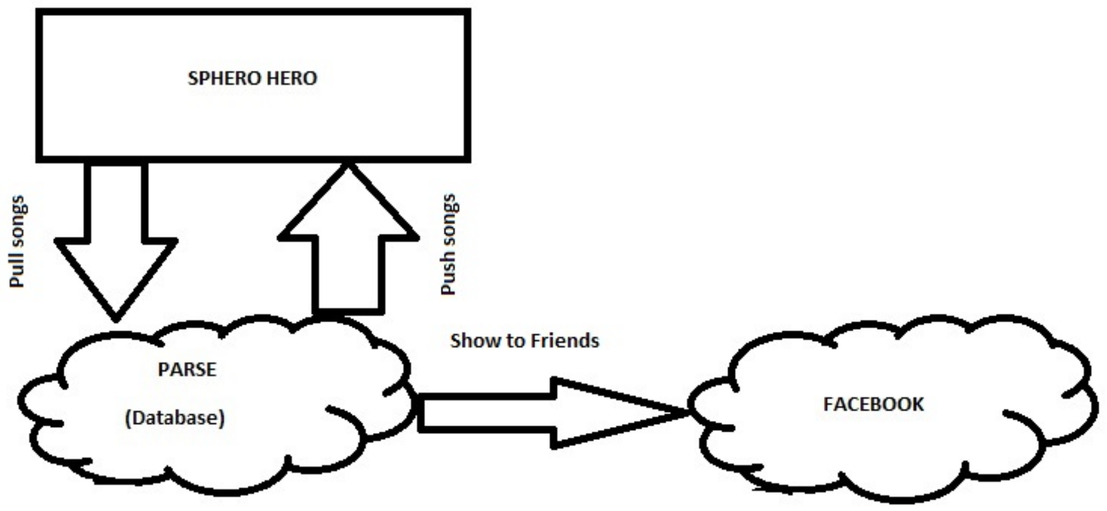

Figure 3: Architecture Diagram for Parse implementation

Technical Design

Several key problems that we faced when beginning our project included: the need to store large amounts of information in a location that could be widely available, be able to access that information within a timely manner using threads, and the creation of an accurate algorithm that would provide constant results when the Sphero is moved by the user.

Making use of Parse required some finesse on our part, because many of us had never used an external API integrated into our code. Another was making use of a remote NoSQL database, and implementing it well. This meant that we had to make as few queries as possible, as well as make it fast. We had to learn to implement blocks by running queries, and how to run them on another thread. This accomplishes a few different tasks: it allows the app to continue completing tasks while querying in the background, and allows us to avoid nasty hang-ups when more data is added to our database.

Another big hurdle our project depended on was correct detection using an algorithm. This was a small problem, when considering that none of us had ever integrated hardware into code, on top of trying to do it on a mobile platform. Starting with Sphero itself, we had to understand how the robot sends packets to the device it's attached to, and how to peel those apart to only get the data we wanted. Figure 4 below shows how we started with individual packet information, as well as a sum of the three, to begin understanding peaks as they came in.

However, after some testing with a very rough first algorithm, we began to realize that some sums were coming in smaller than the directions, and that checking the direction was very tedious. After letting the problem hang in the back of our minds for a bit, we came up with a solution: take the absolute value of each coordinate (x, y, and z), and then sum them. This resulted in the peaks, regardless of direction, always remaining positive. These peaks were easier to analyze because they were much more prominent (instead of summing negatives with positives, we were summing all positive values), and gave us a much easier value to check when looking at the total peak.

Once that was working properly, we faced one final problem when it came to refining the algorithm. Most peaks were very straightforward, but, if the user jerked the Sphero around sporadically, it sometimes created peaks that weren't supposed to be definite moves. So, we analyzed the data - through tests - by passing it through both a high pass and a low pass filter. The low pass helped to smooth out some of these crazier peaks, and provided data that was easy to analyze using our 'sliding window'. Figure 5 shows the difference between the above data that was being analyzed before, and our new, smoother data from the Sphero packets.

It's much more obvious in Figure 5 that the sum (purple line) is defined and relatively constant. We spent a little more time refining down a constant that was high enough to register most movements without picking up every single peak.

Design and Implementation Decisions

Our team made many important design decisions, one of the main being that we decided to make use of the online NoSQL database, Parse.com. This was a decision over the use of "in-house" servers (most likely at Orbotix headquarters) or local storage on the device. Some of the reasoning behind this is that Parse offers an API that allows for cross-platform support between iOS and Android. This connection allows us to share information (scores, songs, moves, etc.), as well as support the same user on both types of devices. Also, barring any Internet connectivity issues, the use of Parse is efficient, as well as free to an extent. After a million Parse requests, costs go up; however, this shouldn't be a problem at first if there is low traffic across the app. Finally, the Parse-provided SDK is easy-to-use, and effective in terms of storing large files like songs, and many objects, like moves.

Another decision we had to make, which depended on our decision to use Parse, was the need to temporarily store songs on the device to access them. We determined that Parse was the most viable option because persisting songs on the device is resource heavy, and likely to not be impermissible for submission. We had to temporarily store songs, however, because of some limitations on the different systems. For Android, the biggest issue was converting the file on Parse back to a .mp3 format. This requires storing the song, converting to the .mp3, playing the song, and then deleting the temp file. For iOS, the issue was that converting a Parse File to NSData (Objective-C's data object) resulted in extra bytes that couldn't be read by the native audio player. So, we had to store the .mp3 locally and reference it by the local URL. As painful as this may seem, it's actually easier than storing everything locally, and much less space consuming.

We also made the decision to make a custom data structure to store incoming Sphero packets when attempting to determine the direction Sphero is moved. In iOS, we implemented a "cliff" by designing a custom double-ended queue using NSMutableArray, that pushes packets on until a designated limit is reached, and then packets start falling off. This is done until a peak is found, and then all packets are dumped and the process restarts. For Android, we created an Array Blocking Queue that uses the First-In First-Out method (very much like the double-ended queue) to push packets on. The reason for using a built in Android structure was issues with originally creating a custom data structure. After some discussion, we decided to move to a built in queue implementation.

After talking with our client, we came to the decision that the app would be much better run in landscape mode. This is advantageous for a few reasons including the fact that the game looks much more professional in landscape mode, and that it's easier on Android programming. The code needs to work on the many different types of screens among Android devices, and therefore it's easier to change for the different screen sizes.

To evaluate accelerometer data from Sphero, we took directional information (x, y, and z) from each packet, took the absolute value of each, and summed them together. What this provides is a much bigger peak when a movement is detected, and a much bigger peak for reviewing packet values. This was a decision we made because our previous method was summing each direction and then taking the absolute value. This provided a different peak that was not as useful for telling us which direction we had just moved the device.

For iOS, we have four view controllers to deal with all of the different handoffs between pages. There are several reasons we decided to do this, including lesser complexity and fewer connections to worry about. With less complexity, we have less code crossing views, and it decouples the code as much as possible. Also, with as few connections as possible, we try to avoid needless code in multiple places, not to mention less code overall.

We use the offsets found by Sphero to maintain detection accuracy, and use it to make sure the user hits the device at the correct time so that scoring is accurate. This requires us to look into the "past" by looking at when the user hit, and comparing the offset to determine the score for that hit, as well as the combo.

Lessons Learned

Working on a project of this scope, and with a team of size seven, we learned quite a bit from each other and our client. One of the biggest things right from the start was the need to include a .gitignore in our project. With two types of projects - both iOS and Android - being committed on the same repository, we mainly needed to add rules to ignore binaries. These caused some major headaches at first, with seven overlapping projects, but it was quickly resolved because of these rules.

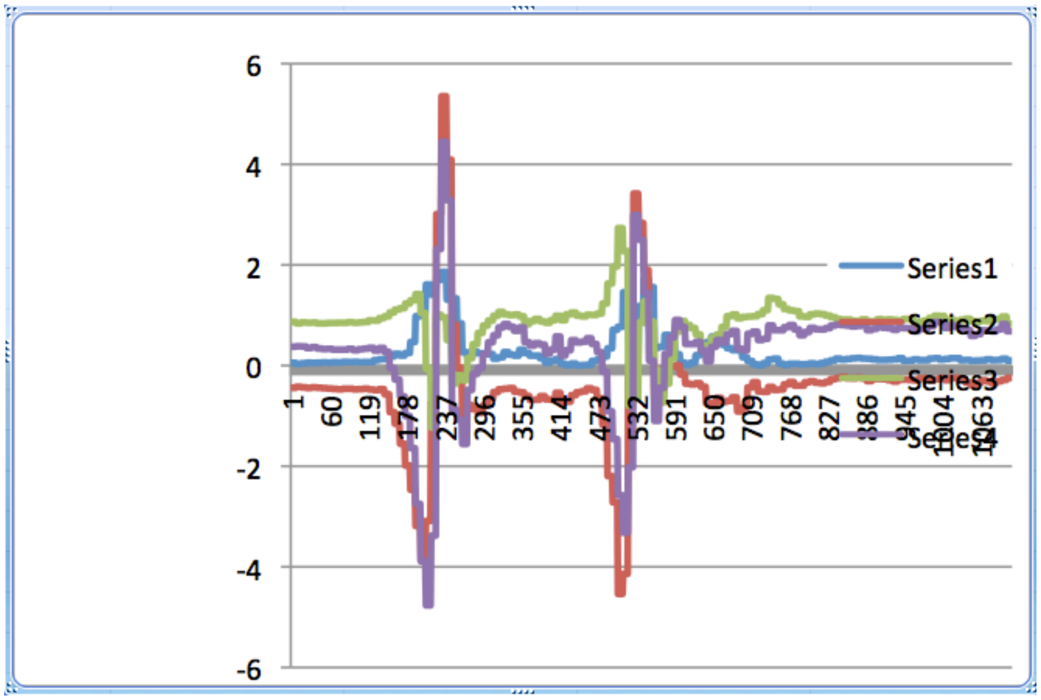

Figure 4: Initial data graph
Blue = X, Red = Y, Green = Z, Purple = sum

Forking on Git, suggested by our client, also proved to be helpful in such a big project. Forking allows the people involved to create an exact clone of the repository, lets them store it in Github under their name, and allow access to it at any time. When any code was committed, it was pushed to the fork instead of the main project, and then any new code could be pulled down. Then, when any of us was ready to include our code in the overall project, we did a small code review, and then integrated in the new code. We were able to pull from that fork, and gave us a second line of defense against merge conflicts, if any arose. This also provided each of us a backup if something went wrong along the way, allowing us to return to a previous commit or reset our copy if major merge errors occurred when pulling from upstream.

One of the best and most effective parts of the Agile process that we did was daily stand-ups. These really helped us focus our day down to what we needed to get done, and pushed us forward when we started to lag. These also gave us perspective when we did sprint reviews, and helped us understand what stories we needed to finish.

The last thing that we struggled with, but managed to overcome, were threads in mobile programming. In terms of mobile devices, performing every function on the main thread can be detrimental to the runtime of an app. This became especially evident when we began implementing Parse query operations alongside our code, as the queries had to be finished before it would continue app processes. What we had to do was block in the background, pushing these queries to complete on another thread. This keeps the app from hanging while a query, like getting the list of songs, processes and returns. This is especially important on a slower Internet connection, as we do not want the app to function poorly. The use of asynchronous programming and notification listeners allow for hardware delay to be accepted, which would also hamper app performance due to Blutooth delay.

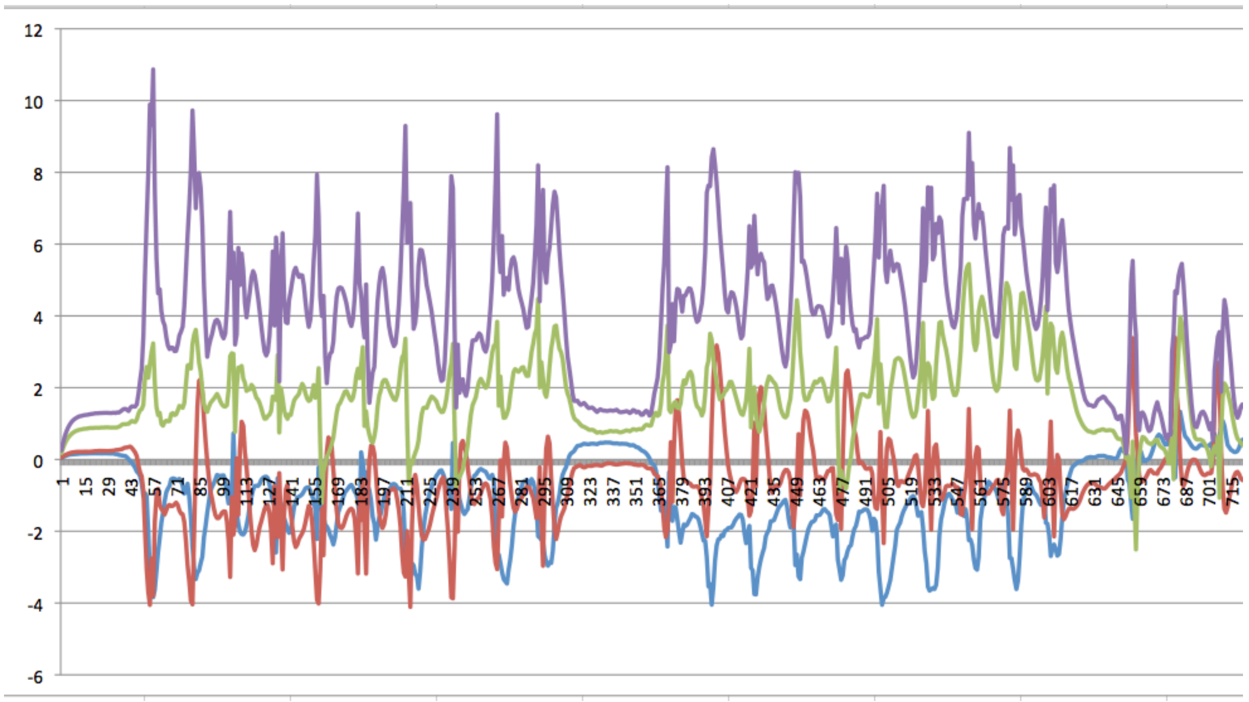

Figure 5: Data after being filtered through a low pass filter

Results

- Playable, professional-looking app
  - App has professionally-made splash screen and art made by the Orbotix Art Department
  - Independent menus designed for easy navigation between pages
- Integration with Parse.com
  - User can either log into the database itself, or use the integrated Facebook login, to store and post scores with other users
  - Also gives the ability for users to store songs made in “Create” mode
- Device correctly determines Sphero direction by using robot accelerometer data
  - Development of an accurate algorithm took several weeks
  - Use of low pass filters and peak finding using sums of accelerometer data
- App works on as many different screen sizes as possible
  - iOS functions on iPhone 4, 4S, and 5
  - Android functions on devices like the Galaxy Note, the HTC Thunderbolt, and the Samsung Tab 2 (Android tablet)
- Didn’t have time to implement a Create section
  - This would allow any user to create tracks using songs from their device and a little bit of time with the UI
  - We had planned it out, but didn’t have time to finish implementation