Yuri Alex Brigance

Portfolio

 Server  Client  Hardware

Professional and personal projects.
One project per page.

10272015

John Deere Harvest Mobile

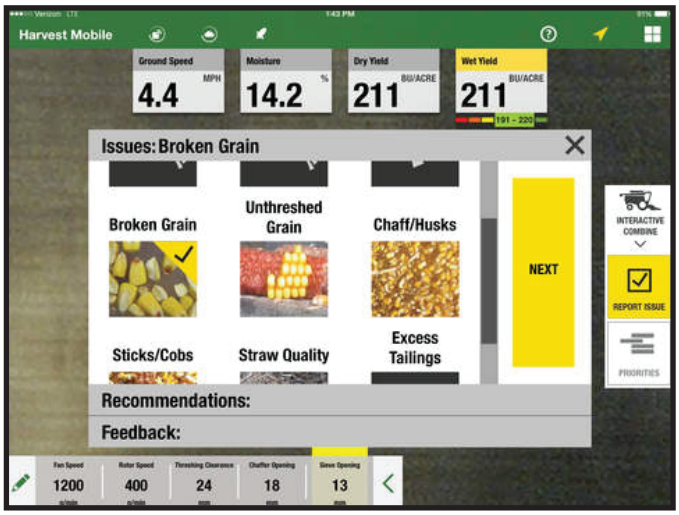
Objective-C, iOS (iPad)

JOHN DEERE

mobile 01

Harvest Mobile is a combine performance monitoring and optimization solution from John Deere that helps growers better understand their data and make machine adjustments to increase performance. Harvest Mobile works with the GreenStar™ 3 2630 Display, compatible S-Series combines, and combine software to enhance the harvesting experience.

Harvest Mobile wirelessly sends the harvest information from the combine to an iPad® inside the cab. The iPad displays the harvest information in maps that make it easier for the grower to understand the yield performance.



Interactive Combine Adjust (ICA) is an intuitive user flow wizard that makes it easy to identify issues and adjust settings to optimize harvesting performance. ICA wirelessly adjusts combine settings from the iPad when the grower approves an optimization recommendation.

This project is not yet in production. As much as I would like to talk about how cool it is, I can't do so until it has been revealed to the public.

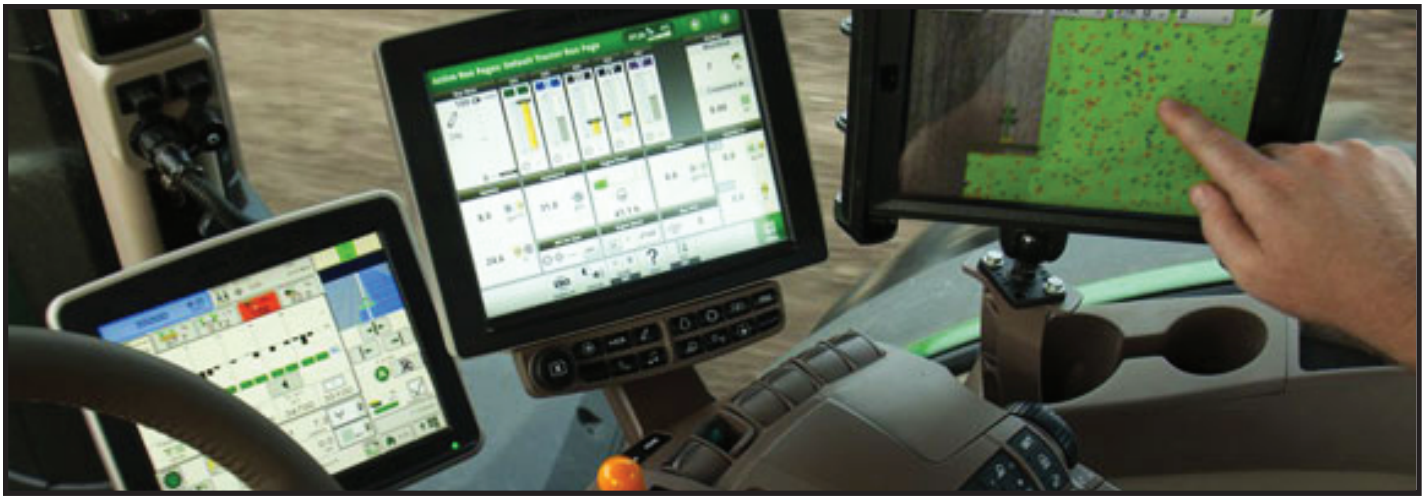
John Deere SeedStar Mobile

Objective-C, iOS (iPad)

JOHN DEERE

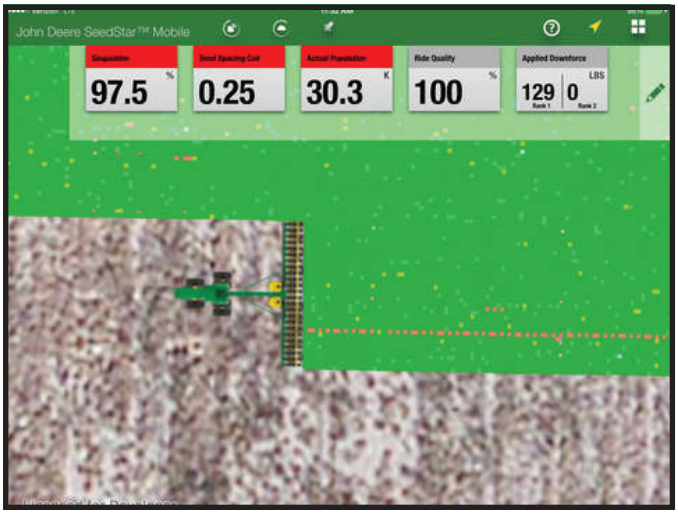
mobile 03

SeedStar™ Mobile is a high definition planter monitoring solution that helps growers better understand their planter performance. SeedStar Mobile wirelessly sends the planting information from the planter to the iPad inside the cab of the tractor. The iPad displays the planter information with high definition, row-by-row maps that make it easier to understand the planter performance.



While in the field, the app can also wirelessly transfer the planting information to the cloud by utilizing the cellular capabilities of the iPad. The data can be utilized for future reference to help growers more efficiently plan and manage their farming operation.

Growers can see fluctuations in singulation as they occur and get the information needed to correct these and other issues, quickly. The dashboard is customizable and a the user can select from a large number of variables, based on what is most important to them.



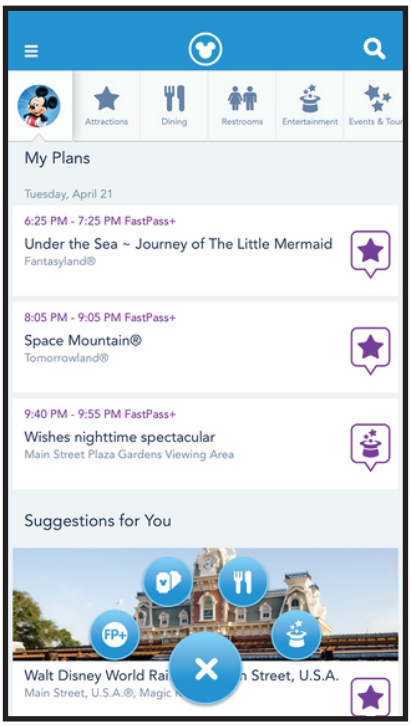
Walt Disney
My Disney Experience

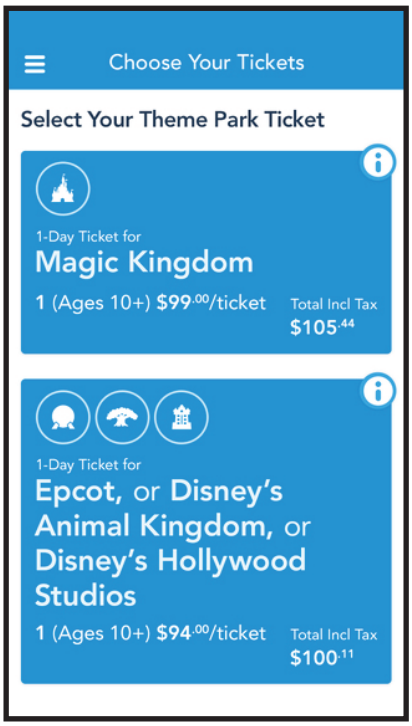
Objective-C (iPhone/iPad)

client

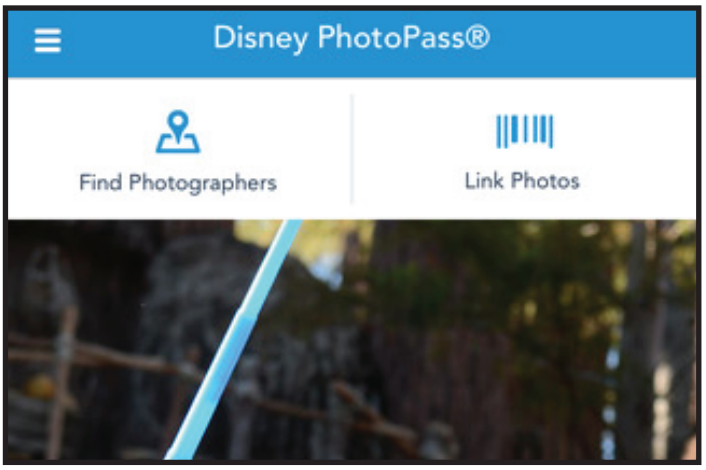
04

My Disney Experience mobile app has an immense amount of features that provide great value to the guests. Users can navigate the park, see wait times, make dinner reservations, get Fast Pass on their mobile device, plan their trip, manage tickets, and much much more. The app looks beautiful too!





I was primarily involved in iOS client-side development, and frequently utilized my knowledge of Disney's back-end server systems, which I acquired during previous engagements.

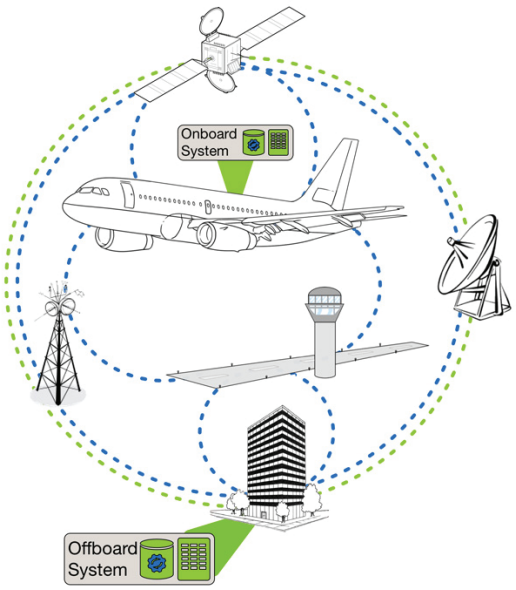


iJet Onboard Data Platform

Java, JavaScript

server, hardware, client 05

The iJet solution begins with the iJet Platform which includes a software package onboard each aircraft connected via IP links to an intelligent ground system. The airborne platform integrates aircraft and ground data with EFB (Electronic Flight Bag) and Cabin devices while applying business logic and analytics to airborne data in real time. The ground system aggregates aircraft data, connects to ground based applications and analytics, integrates to corporate back end systems and moves data up to the aircraft using similar business logic.

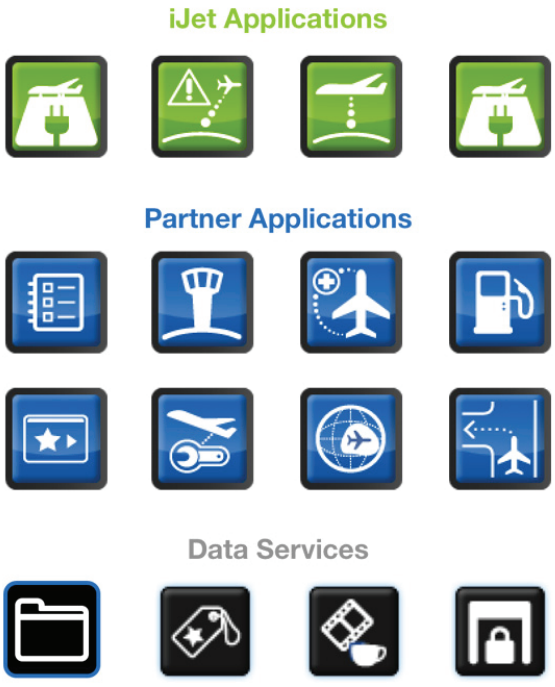


The data is acquired directly from the aircraft telemetry bus (ARINC 429/717) and is parsed, then analyzed in real time by a custom programmable stream processing engine.

The platform then decides which data is important to send offboard to a ground data center based on the condition of the aircraft and which communications link is being utilized (terrestrial or sattelite, for example).

Back on the ground, the data is stored for later analysis. Communication between the aircraft and the offboard data centers is two-way, allowing commands to be sent up to the aircraft in real time.

Pilots can access real-time telemetry, be alerted of various events, and receive commands from the ground mid-flight using any wireless device that can act as an Electronic Flight Bag.



Verizon e-Magazine

Objective-C, iOS (iPad)

mobile

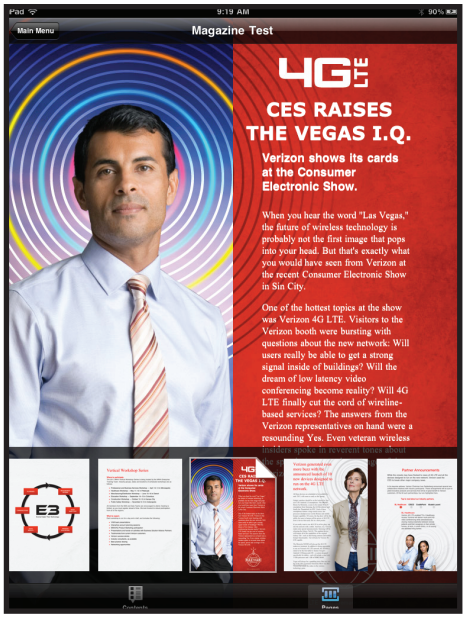
06

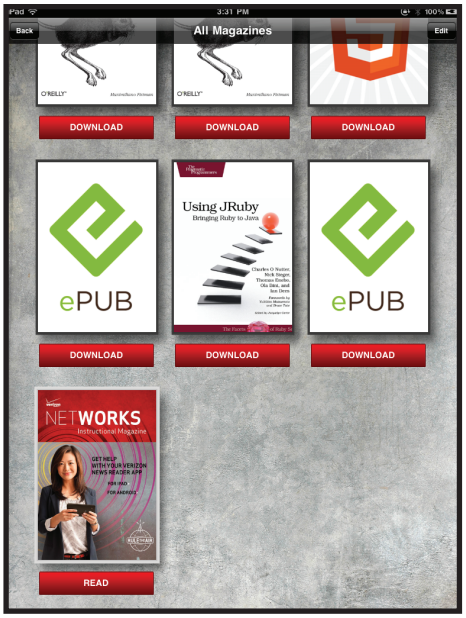
Verizon commissioned this project to use as an internal communication tool for retail employees. The reader had to be fast, download magazine issues from the cloud, and display interactive graphics.

The secondary task was to create a reusable e-magazine framework for use in other applications.

The application relies heavily on multi-threading to speed up performance. Some particularly interesting pieces of code that I had to implement include thumbnail rendering of ePub pages, the ability to inject JavaScript into a UIWebView to detect which element was interacted with on the page, simultaneous download capability (for downloading many ePubs at once) and a lot of custom views.

Upon completion, this application's rendering performance outdid Oprah's own e-magazine app.





UIEvolution e-Magazine Platform

Objective-C (iPad)

UIEVOLUTION

mobile 07

The e-Magazine framework which was developed for Verizon lived on to spawn many different applications, a lot of which are still in the App Store today.

The e-Magazine framework which was developed for Verizon lived on to In my experience, a lot of customers have more than one applications that are similar. For example, both Disney and John Deere have applications which share large portions of code. Which is why I always strive to design all application components to be reusable, and testable in isolation. That way they become reusable modules.

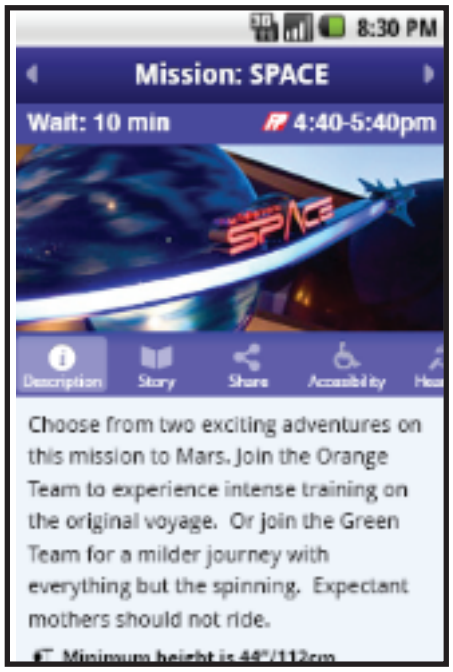
Walt Disney Mobile Magic

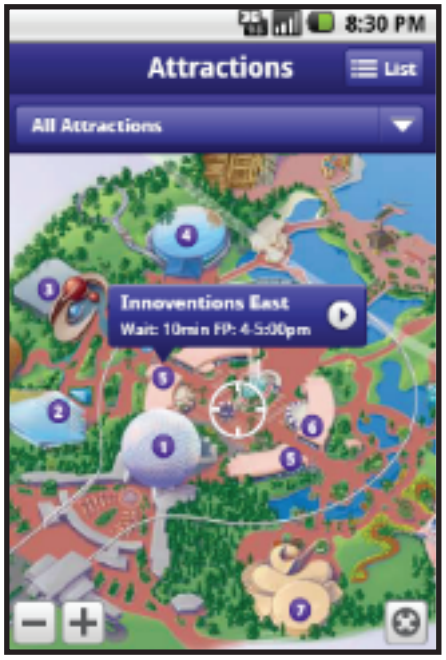
UJML (Cross-Platform)

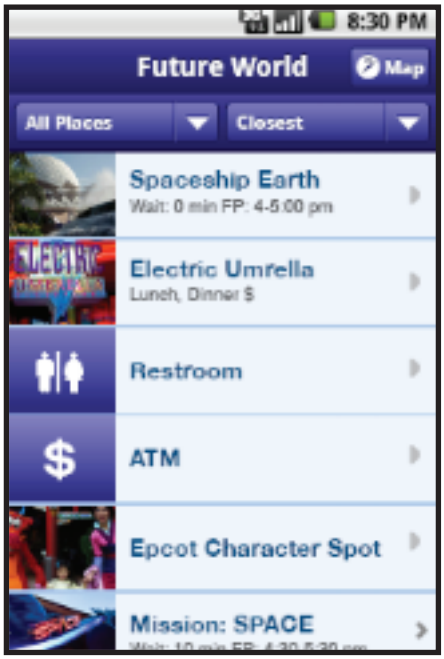
mobile, server

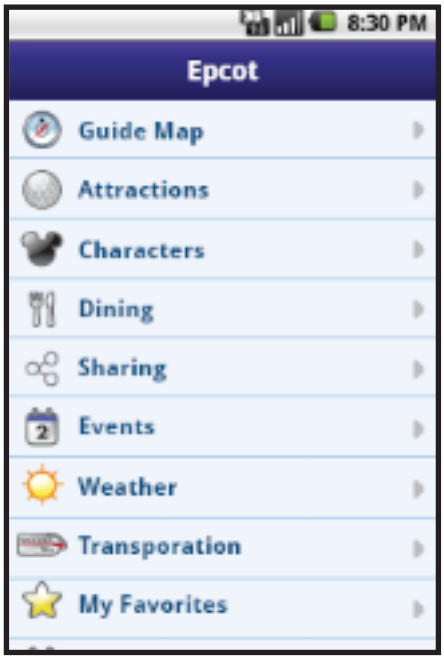
08

Disney Mobile Magic was an application developed using a proprietary cross-platform programming language called UJML. The language was created by a company named UIEvolution. This application ran on many platforms, such as iOS, Android, and Blackberry. Additionally I had a hand in working on the server-side web services for this app.



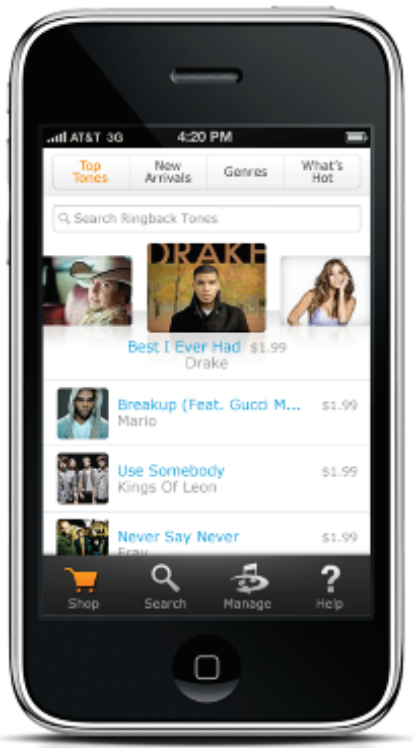


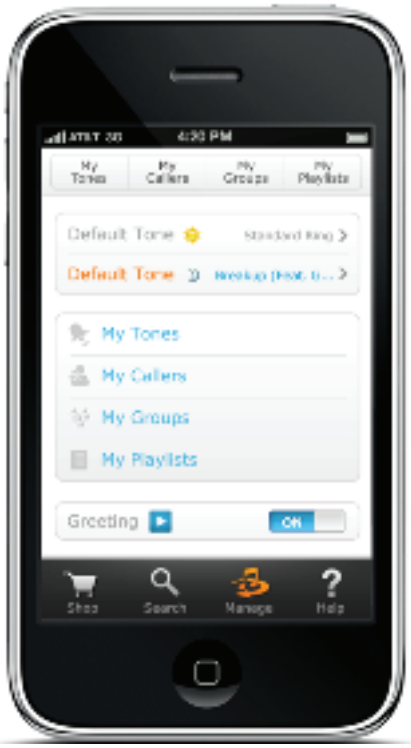


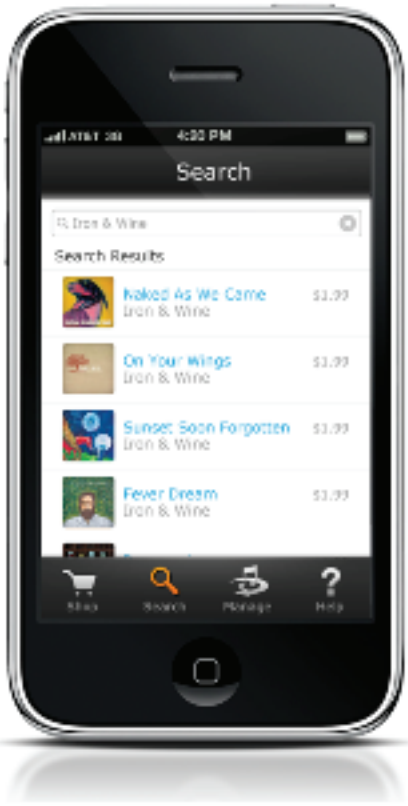


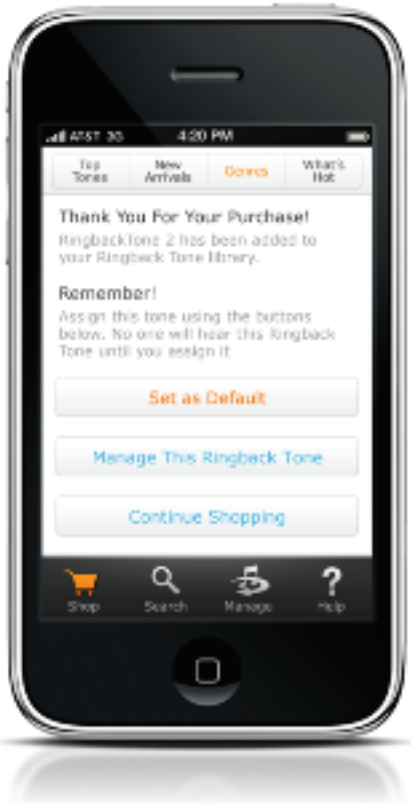
AT&T
Ringback Tones
iOS (iPhone)

mobile 09









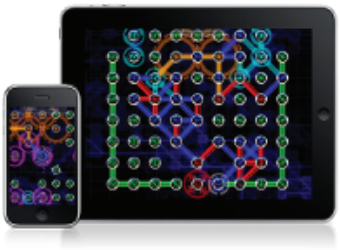
Atomic Cactus
Artilect

iOS (iPhone/iPad)

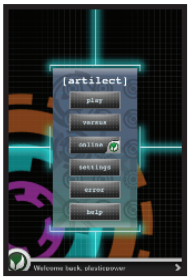
mobile, personal project 10

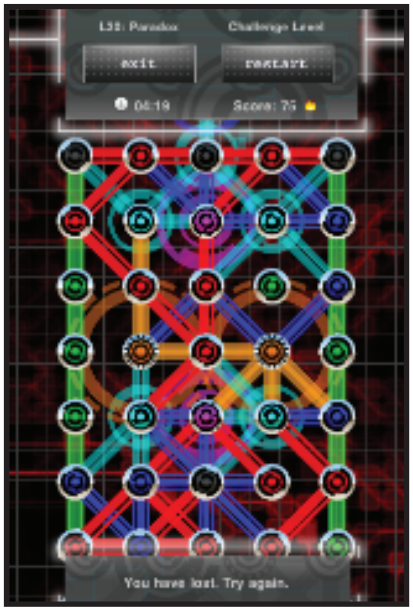
Features:

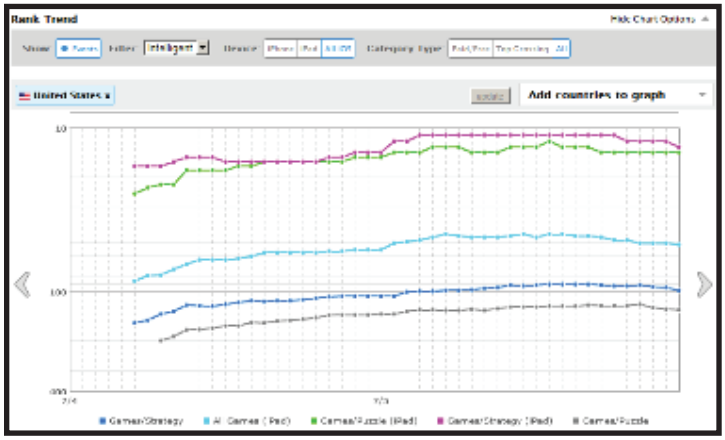
- Over 60 levels
- Online leaderboards
- 2-player versus mode
- iPhone/iPad compatible
- Auto-save & multitasking
- Multitouch controls
- In-game tutorials
- Acheivements



Artilect was a personal project and one of my most successful apps. It was featured by Apple in several App Store categories for several weeks and reached into the Top 15. The application used a homegrown hybrid B*/Dijkstra pathfinding algorithm. Written with OpenGL it featured colorful graphics and particle effects.







artilect

- Top 15 in the App Store
- Featured under “Hot New Games”
- Featured under “New And Noteworthy”
- Recommended by AppSpy

Atomic Cactus
Earthling

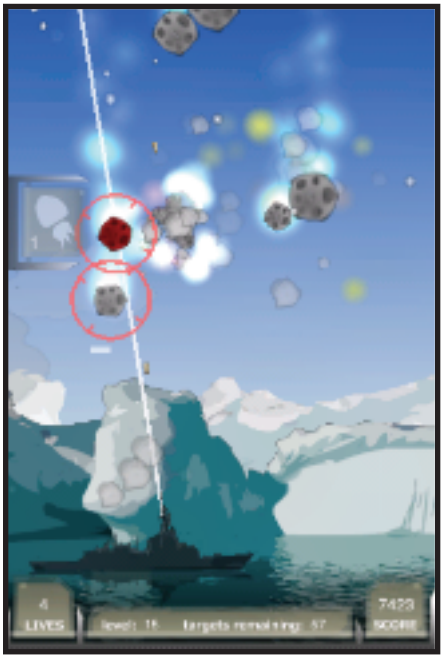
iOS (iPhone)

mobile, personal project 11

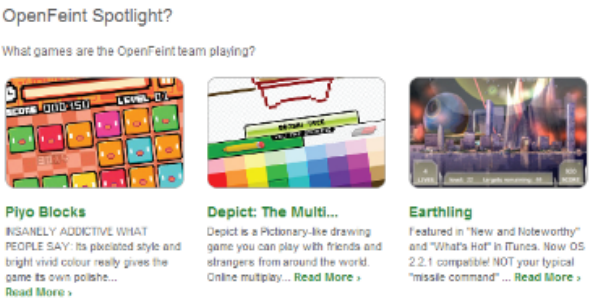
Features:

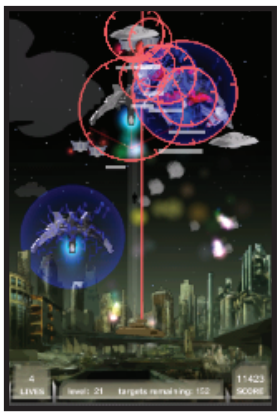
- Over 20 levels
- Online leaderboards
- Over 10 weapons to use
- Multiple powerups
- OpenGL particle effects
- Accelerometer controls
- Achievements
- Story-based gameplay

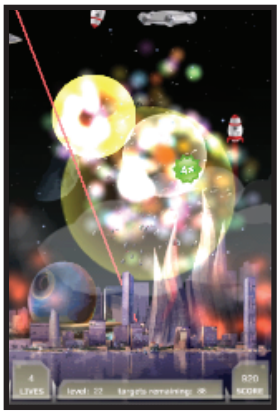
Earthling was my first iPhone game, released in 2009. I knew that I could make a great game if I made something that was very simple to play but was addictive at the same time. I did not want the user to spend countless hours learning how things work. It was therefore decided that the controls will be strictly accelerometer based. To shoot, the user simply had to tap anywhere on the screen. This approach allowed me to utilize the entire screen unobstructed.



- iPhone Apps+ Best Award
- Featured “Hot New Games”
- Featured “New And Noteworthy”
- Reached OpenFeint main page







As with all my personal games and apps, I created all the graphical assets. The game relied on OpenGL and Cocos2D for animation.

Some fun facts:

- It took longer to create the graphical assets than to write the code for this game.

- Developed on the 1st generation iPod Touch using the 1st generation Intel-based MacBook.

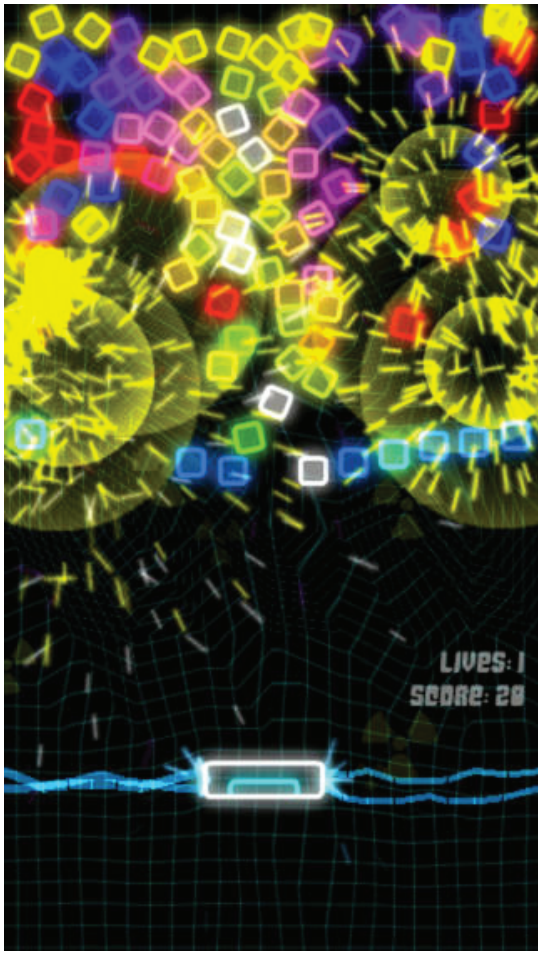
Atomic Cactus
Electroid

Objective-C (iPhone, OSX)

mobile, personal project 12

Features:

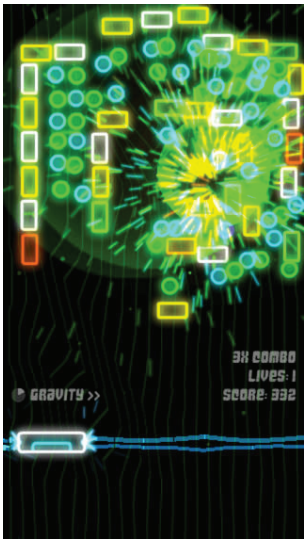
- 12 free levels
- In-app purchases
- OpenGL graphics
- 20+ powerups
- Physics2D engine
- Online leaderboards
- Achievements
- Premium soundtrack

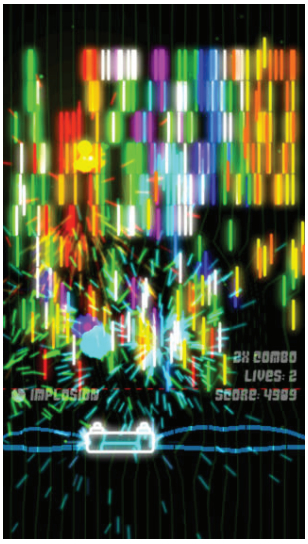


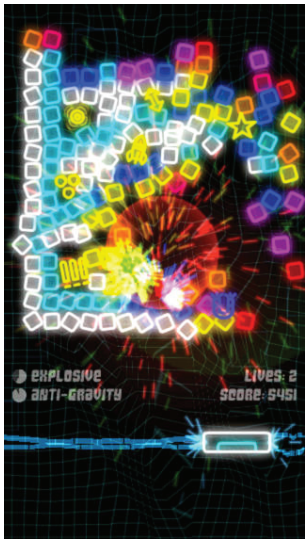
Electroid was my favorite game that I released in the App Store. It featured colorful OpenGL graphics, custom particle effects, intuitive accelerometer and touch controls, lots of levels, and more. It was also an experiment with in-app purchases. There were level packs that users could buy. Unfortunately, the in-app purchase model did not work out, and while the app had thousands of downloads, it did not generate much revenue.

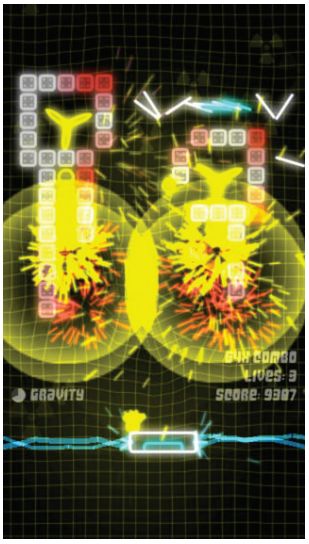
For this game I also created an OSX-based level editor. The level editor was written in Objective-C and worked in conjunction with an existing physics object editor. This allowed me to quickly create and push out level packs.

Fun (or not so fun) fact: I started developing this game way back in 2006, but lost the source code due to hard drive failure. It was eventually re-written and released in 2009, 3 years too late.









University of Washington e-Nutrition

iOS (iPhone), Java

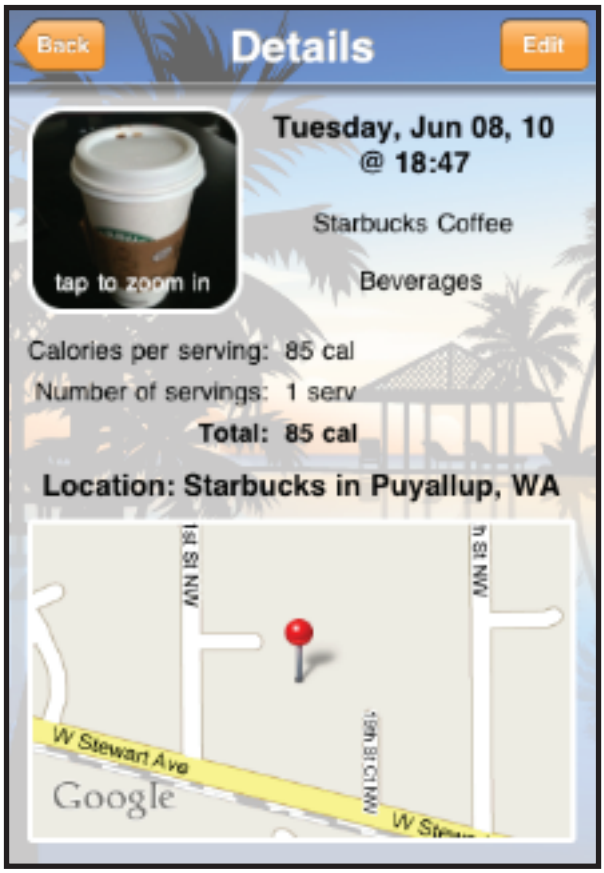
mobile, server

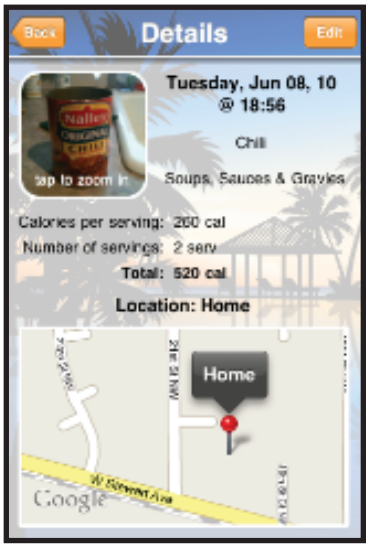
13

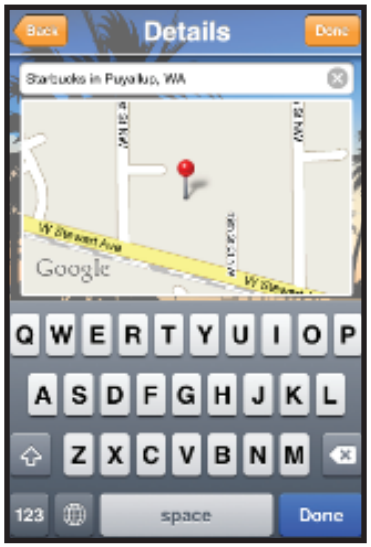
The e-Nutrition app was part of a larger data=mining project at the University Of Washington. The goal of the project was to let people take images of what they eat throughout the day, let them specify the type of food, calorie amount, location, time, etc. This information would be uploaded to a server and used for data mining to extract patterns of food intake, etc.

There was talk about outsourcing the aquired images and descriptions to other branches of the university for image recognition so that we could develop an application where the user simply snaps a picture of a food item and it is recognized by our servers and all the nutritional info is automatically populated.

My role in this project was development of the client application for the iPhone. Additionally I set up a server database and web services required for data upload from the client.







Atomic Cactus
Earthling: Survival

iOS (iPhone)

mobile, personal project 13

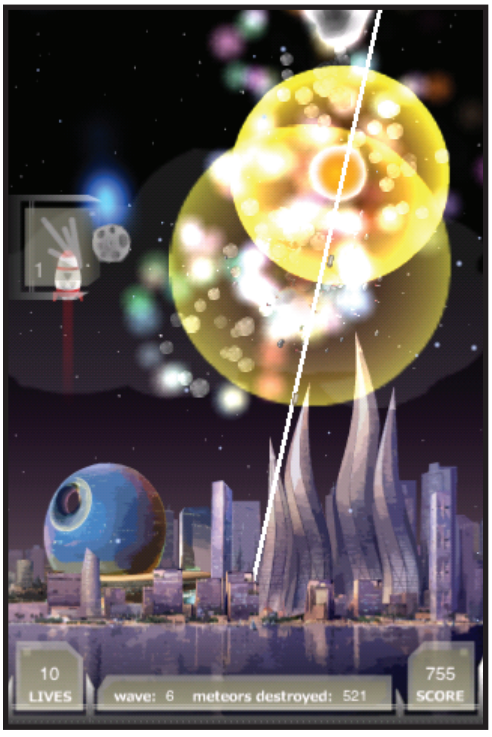
Several months after the release of Earthling, I decided to follow up with a promo game loosely based on the full version. The game was created in under two weeks based entirely on Earthling's existing game engine.

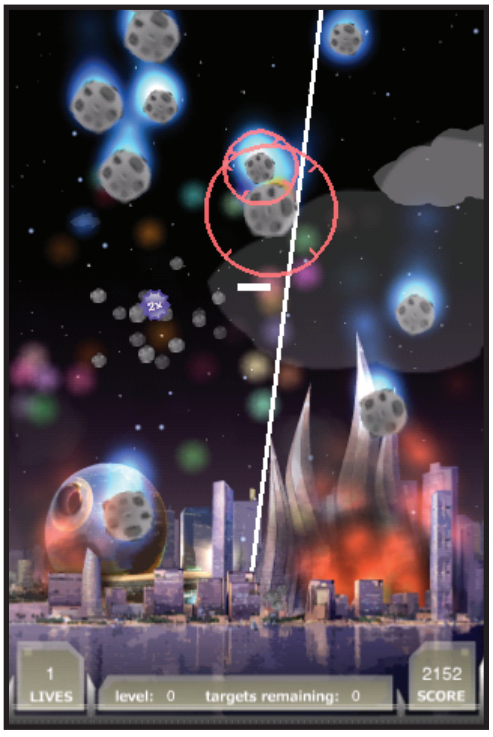
This promo game features a different gameplay (a survival mode) where the player defens a city from an ever increasing salvo of asteroids. Think “Missile Command” but on steroids.

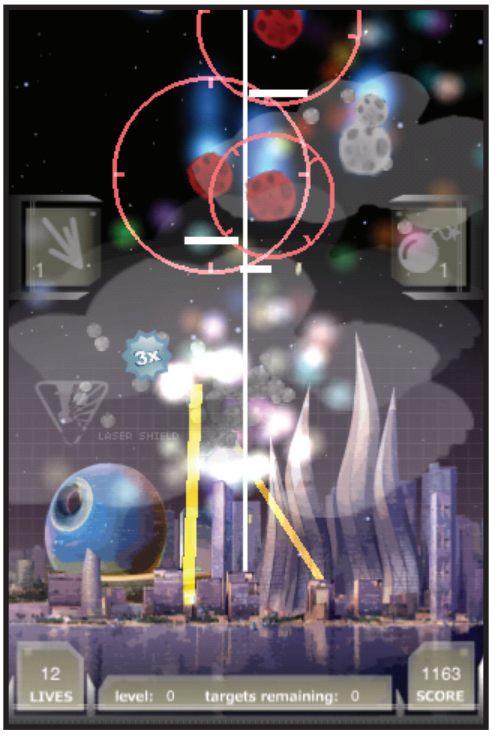
The game also featurs some unique powerups that were different from the full version of the game.

Earthling: Survival was not a demo version, instead it was a standalone fully featured game aimed to promote the more complex story-based Earthling.

As an experiment in app promotion, this did nothing to promote the full version. But the free promo game proved quite popular with lots of players competing online.







Walt Disney DSAA

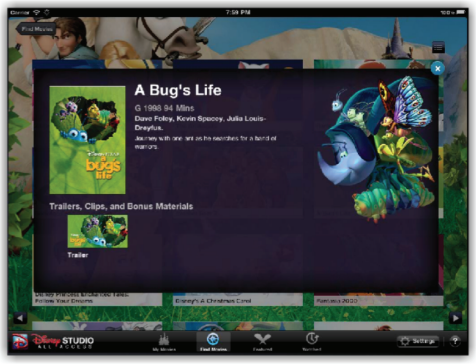
Java, Objective-C

server

14

DSAA stands for Disney Studios All Access. This was a project that allowed users with a Disney.com account to purchase and stream Disney video content.

I was directly involved in creating a suite of RESTful web services for client consumption. These included registration, account management, media catalog, reward points, and more. We used technologies such as Memcached, MongoDB, OAuth, Spring, CXF, JAX-RS. The code was primarily written in Java. I periodically helped a little with the iOS apps as well.



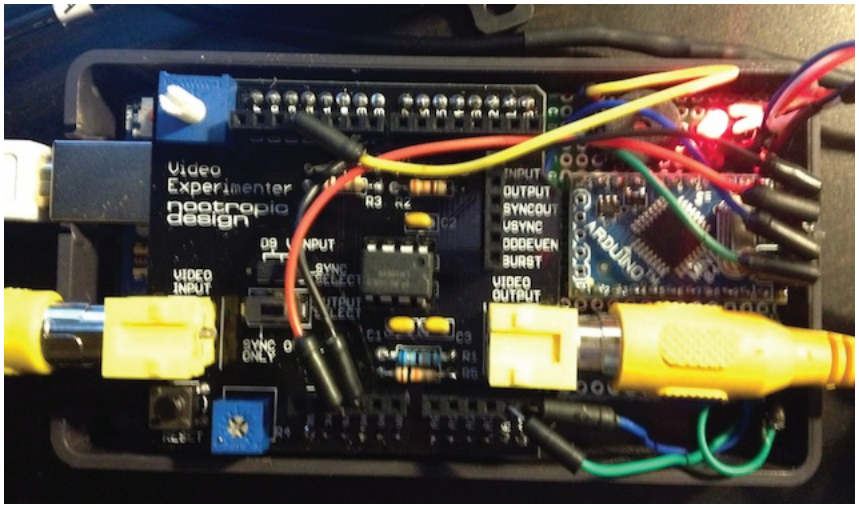
During my work on the web services for DSAA, I developed a unique annotation-based API versioning system. This system allowed developers to annotate API calls within the source code so the server knew which method to execute.

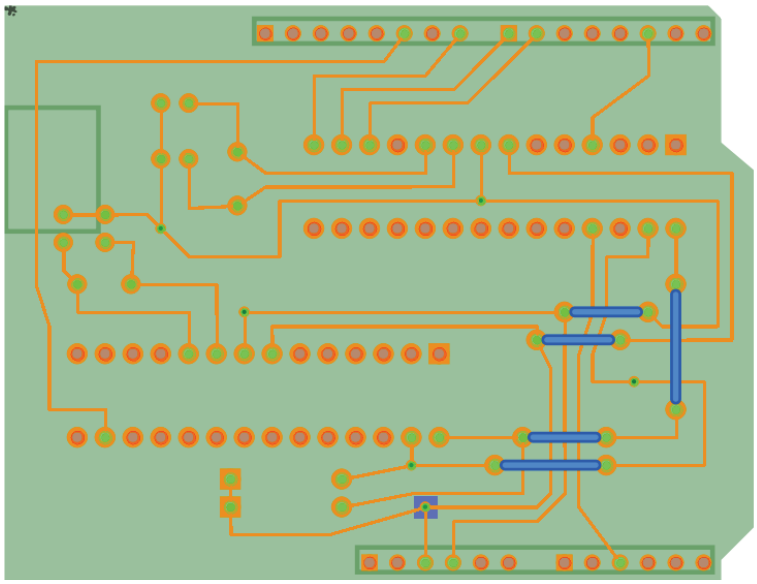
Atomic Cactus
Parkonator

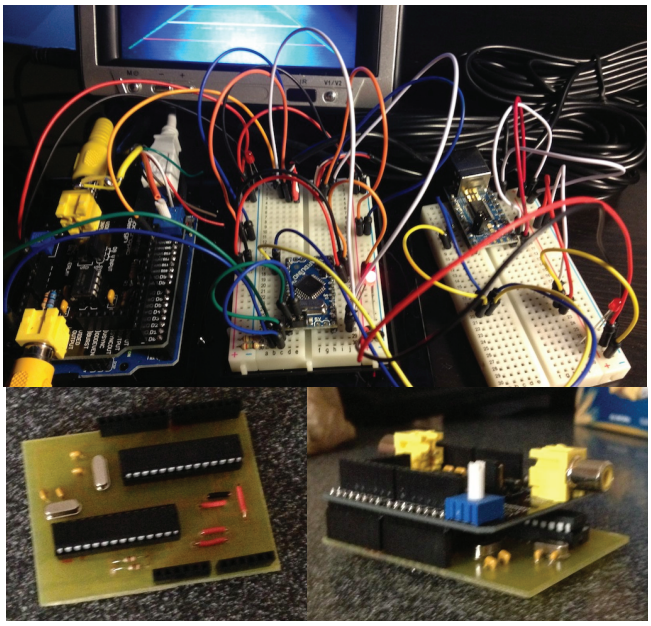
Arduino

Parkonator is an open-source parking-assist system that I designed using the Arduino platform. It uses ultrasound sensors to detect distance from objects. This information is then digitally overlayed over an RCA backup camera and piped into any compatible video display in the vehicle's dash.

Parkonator has so far survived 5 years inside my Pontiac GTO without issues.







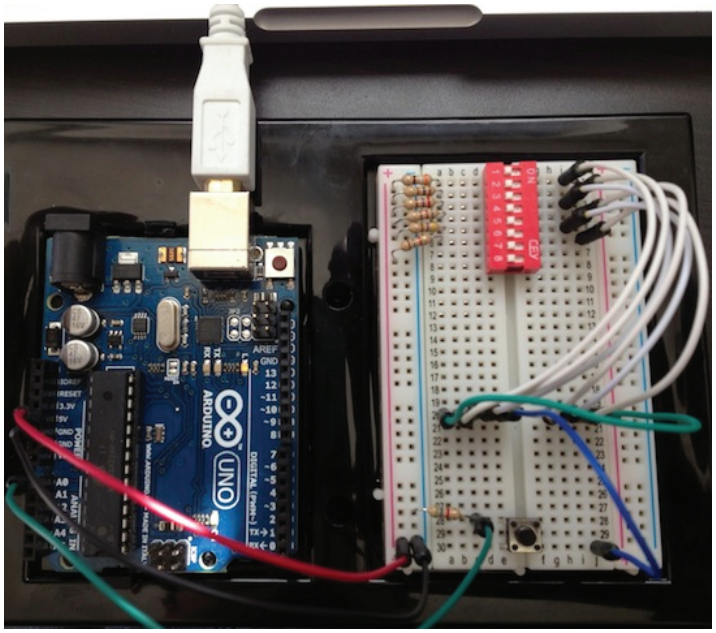
Atomic Cactus
Steering Wheel Remote

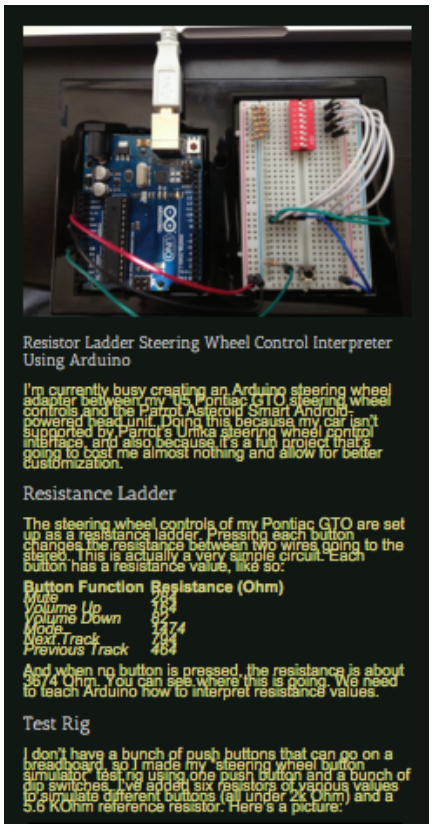
Arduino

Another open-source project of mine was a steering wheel adapter for all resistor-ladder based vehicles.

This adapter allowed the user to connect their existing steering wheel controls to an aftermarket stereo (Android-based) on the cheap, without spending over $150 on a device that can be built at home for under $30.

I used an Arduino Uno R3 and a simple home-made ohm meter circuit to detect which button was pressed. For Android head units, the Arduino was them flashed with special firmware that made it act as an HID device.





Resistor Ladder Steering Wheel Control Interpreter Using Arduino

I'm currently busy creating an Arduino steering wheel adapter between my '05 Pontiac GTO steering wheel controls and the Parrot Asteroid Smart Android-powered head unit. Doing this because my car isn't supported by Parrot's Umka steering wheel control interface, and also because it's a fun project that's going to cost me almost nothing and allow for better customization.

Resistance Ladder

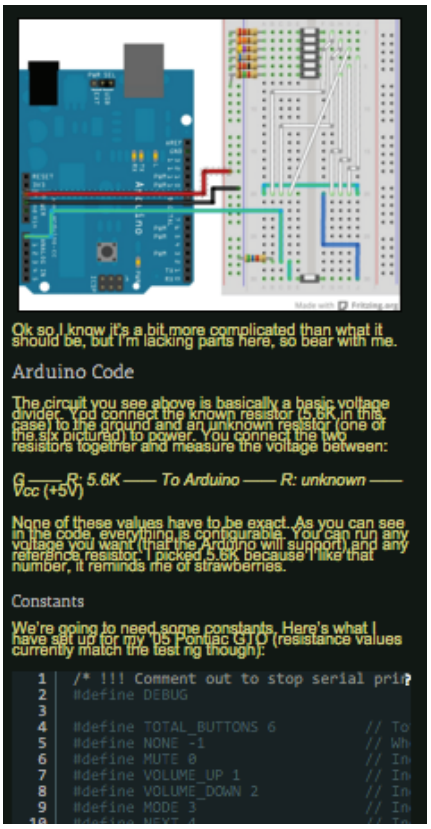
The steering wheel controls of my Pontiac GTO are set up as a resistance ladder. Pressing each button changes the resistance between two wires going to the stereo. This is actually a very simple circuit. Each button has a resistance value, like so:

Button Function	Resistance (Ohm)
Mute	183
Volume Up	283
Volume Down	92
Mode	1374
Next Track	743
Previous Track	484

And when no button is pressed, the resistance is about 3674 Ohm. You can see where this is going. We need to teach Arduino how to interpret resistance values.

Test Rig

I don't have a bunch of push buttons that can go on a breadboard, so I made my steering wheel button simulator test rig using one push button and a bunch of dip switches. I've added six resistors of various values to simulate different buttons (all under 2k Ohm) and a 5.6 KOhm reference resistor. Here's a picture:



Ok so I know it's a bit more complicated than what it should be, but I'm lacking parts here, so bear with me.

Arduino Code

The circuit you see above is basically a basic voltage divider. You connect the known resistor (5.6K in this case) to the ground and an unknown resistor (one of the six pictured) to power. You connect the two resistors together and measure the voltage between:

G ——R: 5.6K—— To Arduino —— R: unknown —— Vcc (+5V)

None of these values have to be exact. As you can see in the code, everything is configurable. You can run any voltage you want (that the Arduino will support) and any reference resistor. I picked 5.6K because I like that number, it reminds me of strawberries.

Constants

We're going to need some constants. Here's what I have set up for my '05 Pontiac GTO (resistance values currently match the test rig though):

```
/* !!! Comment out to stop serial prin
#define DEBUG

#define TOTAL_BUTTONS 6           // To
#define NONE -1                   // Wh
#define MUTE 0                    // In
#define VOLUME_UP 1               // In
#define VOLUME_DOWN 2             // In
#define MODE 3                    // In
```

```
#define VOLUME_DOWN 2             // In
#define MODE 3                    // In
#define NEXT 4                    // In
#define PREVIOUS 5                // In

#define TOLERANCE_PERCENT 10.f    // M
#define RESISTANCE_PIN 0          // A
#define R_KNOWN 560.f             // T
float VOLTS_IN = 5.f;             // V

/* Button resistance values and indexe
float BUTTONS[TOTAL_BUTTONS] = {390.f,
```

What's configurable here? Well, you can change your reference resistor value, your voltage (which will probably always be +5V), button resistance value, number of buttons, which buttons represent what function and the tolerance percentage. The last one (tolerance) determines how close the current measurement has to come to a pre-defined steering wheel control resistance value before being considered a button press.

Variables

These will change over time:

```
int currentButton = NONE;         // Cur
int rawVolts = 0;                 // The
float voltsOut = 0.f;             // Vol
float resistance = 0.f;           // Unk
```

Here we store the currently selected button, current resistance, voltage, etc.

Measuring Resistance

Define a function that will convert the input voltage to resistance (and a function that prints it):

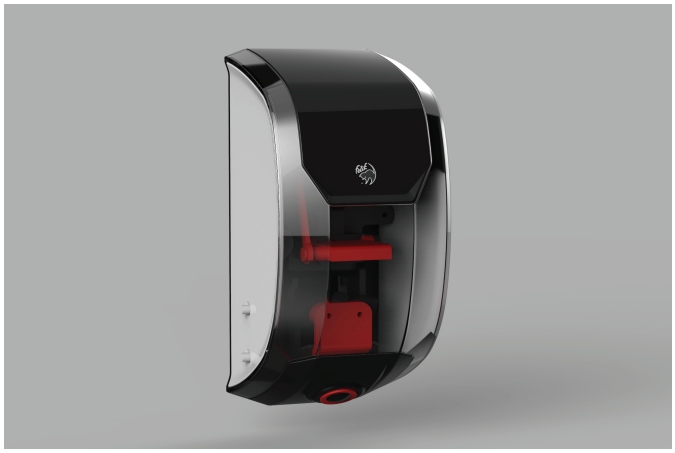
```
// Calculates the resistance.
void calculateResistance() {
rawVolts = analogRead(RESISTANCE_PIN);
voltsOut = (VOLTS_IN/1024.0) * float(r
resistance = R_KNOWN*((VOLTS_IN/voltsO
}

// Prints the resistance (if DEBUG is
void printResistance() {
#ifdef DEBUG
Serial.print("Voltage: ");
Serial.print(voltsOut);
Serial.print(", Resistance: ");
Serial.println(resistance);
#endif
}
```

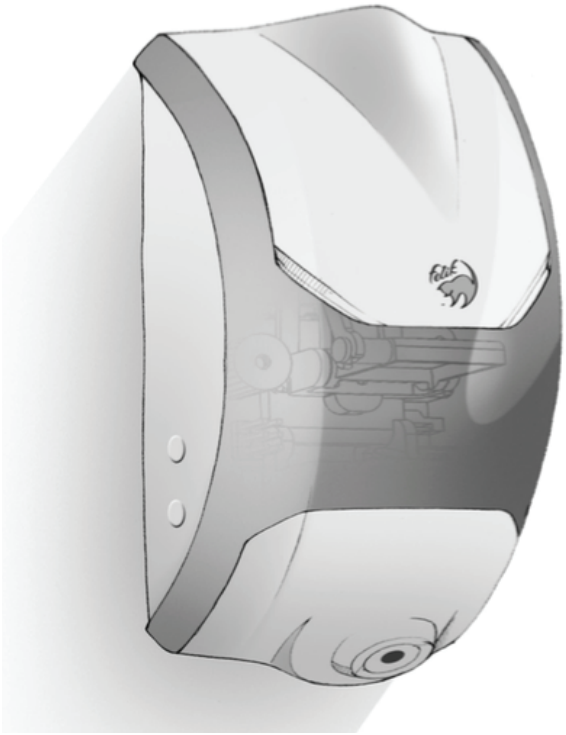
Angry Lychee Felik

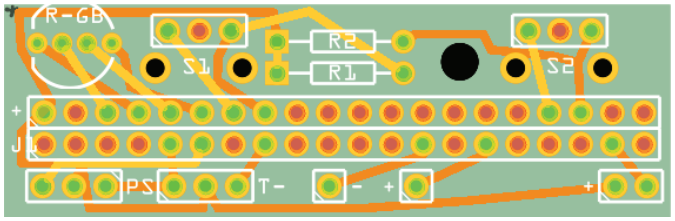
RaspberryPi, Linux, Java

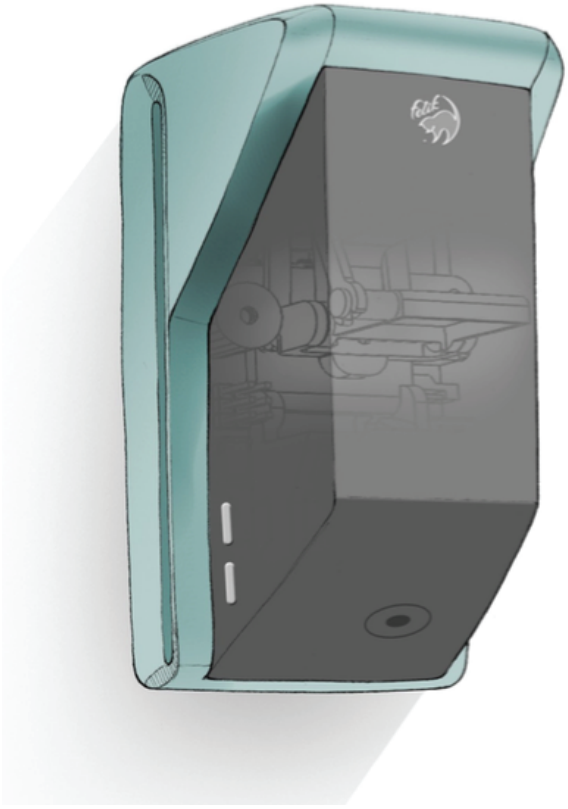
hardware, personal project 17

Felik is my first commercial hardware project. I can't talk a lot about what it does, because it will be launching on KickStarter soon, and I don't want to spoil the surprize. For now, I'll include these images to show what it will look like without going too much into detail about what it does.









Atomic Cactus PocketWin

Flash ActionScript, C# (Windows Mobile 2003)

mobile, personal project 18

This was my first commercial mobile application. Back in 2003, it was developed for Windows Mobile 2003 PDAs. There weren't any smartphones around back then. The application subsequently stayed in the Top 10 best selling app spot on two largest mobile app websites at the time: Pocket Gear and Handango.

The concept behind this app was to create a skinnable application launcher that would mimic various existing deskrop operating systems of the time. It came with a default Windows XP skin, and contained many more. Users could extend the application with plugins. The Android platform currently allows the user to completely replace their app launcher, and this app was the granddaddy of them all. The app was written in Flash ActionScript and C#, and ran on a large variety of devices due to Flash's cross-platform nature.