Making Friends (out of thin air)

Hello, I’m Zach! My pronouns are he/him.

This is Dani!

<Dani introduces themself>

Slide: What is a Tamagotchi?

A bit of background in case you are not familiar with Tamagotchis. They are a virtual digital pet. They were a big deal in the early 2000s. My relationship with them at that time was that my mom wouldn’t let me get one. They are made by Bandai, which is the same company that made Digimon.

I’m actually not that well versed in Tamagotchi lore, so this is pretty specifically focused on this one version of the Tamagotchi, and just the Infrared part of it. If you know more about other types of Tamagotchi I would love to talk to you! I know there’s a deep well to tap there and a lot of it is in Japanese, which I cannot read.

The ones I’m talking about today are the 2024 20th anniversary re-release of the Tamagotchi Connection version 3. There have also been a variety of other versions of Tamagotchis Connection and otherwise, but they use a different IR format, and I haven’t spent the time to figure it out yet. If this sounds fun and you have an old tamagotchi laying around, please get in touch!

Slide: Why?

I took an interest in weird Infrared stuff when working with Dani on reverse engineering Pixmob wristbands. They will tell you a bit about that project now.

< Dani talks about Pixmob>

While spinning our wheels on that, we learned a lot about different infrared signals, different ways to modulate and encode data, and ways to find OSINT about products with proprietary formats.

Slide: Brief tour of Infrared data transfer

Now you're about to get the world's quickest, dirtiest, shortest explanation of digital signals and how they relate to infrared.

https://pravin-hub-rgb.github.io/BCA/resources/sem3/datatbc301/unit2/index.html

Slide: Encoding and Modulating data

In the computer there is some information. That could be text, numbers, images, whatever. It is all represented as numbers, which are represented in binary as ones and zeros.

To get that information into the physical analog world, we need to represent two different states for 1 and 0. This usually means making a wave do two different things. The wave could be light, sound, radio, or something else probably.

When it does one thing, that will mean "One" and the other thing will mean "Zero". In our case for infrared, the two things will only involve turning the light on and off.

Doing that, getting the digital 1s and 0s into the analog world is (approximately) called "modulation". Reversing the process to get the data back out is "demodulation". That's where the word "modem" (possibly the most misused word in operational IT) comes from. Modulate-demodulate.

The Tamagotchis, and most IR signals I've encountered, are not actually doing anything very interesting with the analog world. So from now on, we're basically going to act like it's digital and ignore it. For our purposes it is just on or off. Nothing crazy with the phase, or the frequency, or

anything other than “on or off” for the amplitude. If this sounds intriguing, go talk to someone who is into HAM radio because they do all the funky modulation stuff.

Slide: Carrier Waves

Before we totally turn our backs on the analog world, but I wanted to include a quick note about carrier waves, because if you’re buying an Infrared receiver or transmitter, it’s useful to know.

Pretty much all IR remotes use a carrier wave, meaning that if their light is “on” it’s actually blinking really fast, usually around 38 to 40 kHz.

Why? Well, there is hardware that can filter out anything else that’s not blinking at that frequency. This helps with interference from constant sources of IR like the sun, or IR flood lights for security cameras.

You must handle the blinking somehow, either in hardware or software. Sending (illuminating the LED) is easy to handle in software. Most microcontrollers have a PWM (pulse width modulation, more on this later) PWM peripheral that lets you easily blink the LED at whatever frequency you want. So it’s usually fine to just get a regular old IR LED, and you can make it work for a carrier wave and for no carrier wave.

Receiving a signal, so if someone else is blinking an LED at you and you want to decode it, is harder to adjust in software. It’s more important to get the right receiver. Most things called “IR receiver” expect a carrier wave. They reject any other frequencies and hand the microcontroller a constant high or low voltage. However. If you need to accept an IR signal with no carrier wave, or one with a different

frequency of carrier wave, you're out of luck. The software literally can't see it because the hardware is filtering it out.

It is possible, but maybe not advisable, to go the other way and use an analog IR receiver to receive a signal that uses a carrier wave. I made something that does that on last year's def con badge, which has analog IR hardware for no carrier wave, and it kind of works. Please come find me after this if you want more information!

To summarize, none of the carrier wave stuff matters for the rest of this talk. Just pretend the light is solidly on or off.

Slide: Encoding and Modulating Data

Back to the question: How can we represent 1s and 0s by turning an LED on and off?

Slide: On Off Keying

The easiest way is probably this. On off keying- 1 is when the light is on for some length of time, 0 is when it is off for that length of time. This is actually pretty uncommon, because it turns out decoupling the data from its physical representation is very useful.

Slide: Logical/Physical

One concern when designing an IR encoding is how much time the LED spends on. It will last longer and use less power if that can be minimized. If sending 80 gigabytes of 0xFFFFFFFFF burns out your LED and runs down your battery, that's not exactly ideal. This is the duty cycle of the LED—what percentage of the time it's actually illuminated.

Another concern is how long there is between state transitions. State transitions being, LED changes from off to on, or on to off. This is an issue because there's no explicit clock signal, and the transmitter

and receiver need to sync up and agree how long a bit should last. If sending 10 million “0” bits means it goes 5 minutes with the LED just off, how do you know they’re zeros and not just nothing?

Slide: Logical !- physical

So, if the physical and logical worlds are going to be different, it’s useful to have different terms for them. In the logical world I’ll say logical 1, logical 0, or maybe just 1 and 0. In the physical world, if the light is on we’ll call that a mark or a pulse, and if it’s off we’ll call that a space.

Slide: protocols

This is a representative sample of ways that remotes represent 1s and 0s. They’re different, but they all address the issues with on off keying.

NEC uses pulse distance modulation, meaning that the distance between pulses is what changes to represent the two states. 1 is a long gap, and 0 is a short gap.

The extra mark at the end here is just so the length of the space is defined.

And the rest of this over here I included so you can see the overall shape of the code. Excluding the really long one at the beginning, the marks are all the same width, and the gaps vary. And you can see why they’re called marks and spaces, because the marks have a mark on them and the spaces are spaces. NEC also does some stuff where it sends the inverse of each byte. Don’t worry too much about that yet.

Since there's a guaranteed mark and space for every bit, there will never be long gaps between state transitions. And the mark takes up half or less of each bit, so overall the LED will probably have a lower duty cycle. The worst case is all 0s and it's 50% on, but 50% is still way better than all 1s being 100% on with on off keying.

Next one! There are two ways to think about Phillips RC5: One is that it uses pulse position modulation, meaning in a given frame, if the mark is at the beginning, it's 0. If the mark is at the end, it's a 1.

The other way is to think of it is a transition from high to low, or a transition from low to high. This representation is also called Manchester coding, and it is also used in other places, like Ethernet.

Sony uses pulse width modulation, which is probably more famous for its other applications. Pulse width modulation is what you use when you want to dim an LED. If the LED turns on and off fast enough, our eyes think it's just on and see the average of the time spent on and time spent off. You can also use it to approximate sound waves since if it changes fast enough, we just hear the average of the amplitude.

To represent bits, though, this code just uses a longer pulse for 1, and a shorter pulse for 0.

Slide: IR venn diagram cell

To put all of this in context, those protocols all fall into the category "Consumer IR". Everything we've discussed and most IR signals you'll encounter in 2025 belong in that category.

Slide: add tamagotchi

And, in fact, so does the tamagotchi connection.

Slide: add IR category

But there is more!

Slide: add IrDA

IrDA, Infrared Data Association, was a standards organization in the late 90s and early 2000s. It was a Bluetooth competitor, and is sadly pretty much dead now.

SIR stands for Serial IR and got backronymed to Slow IR because they also made Medium IR, Fast IR, and Very Fast IR. These protocols don’t use a carrier wave, so you can’t interface with IrDA using hardware that expects a carrier wave. If you could, this talk would be about an implementation of IrDA for the Flipper Zero.

There is a different type of Tamagotchi, the Tamagotchi iD, that uses IrDA. You can download items onto your computer and then transfer them over to the Tamagotchi with an IrDA adapter. IrDA also used to be built into cell phones sometimes.

I haven’t actually used a Tamagotchi iD, and it’s kind of hard to get your hands on an IrDA USB adapter or cell phone at this point. I’ve got a project on the back burner to make my own IrDA USB adapter, but it is very much not finished.

You may also have encountered IrDA in Palm pilots. They used it for beaming files, if you ever had one of those. And last year’s DEF CON badge has IrDA hardware.

Anyway!

Slide: Many Tamagotchis were harmed

At this point I want to mention Natalie Silvanovich and her talks about hacking Tamagotchis at CCC. When I came across those, and they saved me a lot of aimless wandering. She went more in depth and actually took a bunch of tamagotchis apart, and did really cool stuff with the hardware.

Slide: pictures

These are pictures of some of the things she did. One was simulating the ROM using an arduino so she could make it hold arbitrary data. She also decapped a chip by dissolving it in acid. Go watch her talks!

She mentioned that the IR format is “Nearly NEC” at the end of one of those talks, and fortunately that was still true for the version I had.

Slide: Tamagotchi

So here it is!

You can see the general shape of it sure looks like NEC. It’s got a similar preamble or leader code to signal the beginning of a message. It also has one extra mark at the end just so the last space is well defined.

1 is a mark and a long space, 0 is a mark and a short space. I took some examples I recorded and looked at the lengths of marks and spaces to figure out about what they should be, and this is where it ended up.

I should say a bit about how I was actually measuring this. The Flipper didn’t work because of the way it detects the ends of messages. It timed out in the middle of a conversation, so I could only really get the first one.

I used a raspberry pi pico and some IR hardware left over from the Pixmob experimentation to record a bunch of messages from the tamagotchis, and found that I could replay them and conduct a visit.

Side note, Micropython is awesome. Take a look at it next time you're considering doing something with an arduino or other microcontroller. I really liked it for playing around and prototyping, and you can also make actual serious firmware with it.

I had a problem though, which is that my hardware was pretty janky. Dani heard me complain one too many times about that and made this really cool SAO, and they can tell you a bit about that!

~~~~

Slide: Now what?

To recap, I've recorded a bunch of tamagotchi interactions, I have the messages translated into binary. I've also found that each message is 160 bits long, and each interaction is four messages long. Now what?

Slide: OSI Model

It's DEF CON, there must be some networking folks here and I know you love the OSI model. Until now we've been concerned entirely with the physical layer. It is time to move up a bit.

Sike, this didn't really fit well in the OSI model. I decided to call my layers the message layer and the conversation layer.

Slide: Message layer
~~~~

The message layer is concerned with the format and contents of one message. It has things like the tamagotchi's name, its appearance, and some messages have unique information like which gift is being sent in the fourth message. There is also a 1 byte checksum at the end of each message

Slide: Conversation layer

The conversation layer is concerned with what the four messages mean all together. The conversation type (visit, gift, game) changes a byte in a different way in each message. The fact that the structure is four messages back and forth is also part of this layer.

I see the parallels with TCP handshakes, but I didn't come up with creative names for the messages. They're just message 1, message 2, message 3, message 4.

Also, thank Dani for the suggestion to make these look like chat message bubbles.

Slide: Patent 8,545,324

When I was reading everything I could get my hands on about Pixmob, there were several interesting patents, but they were pretty vague about the data format and implementation details. I hadn't even bothered to look up the tamagotchi connection patent because I thought it would be equally useless.

In fact, it was extremely helpful. It has a very detailed explanation of almost everything you could want to know about your Tamagotchi. What time of day does it brush its teeth? Which byte in a message is the checksum? State machine diagrams for the interactions? Hardware diagrams? Yes yes yes and yes.

Slide: Figures 15 and 25

The official descriptions of these figures because I think patent language is hilarious:

Figure 15 shows a schedule for a day of the virtual living object as an adult in present invention.

Figure 25 is a flow chart of the "initial communication" processing steps of the communication game device of the present invention

Slide: Figures 32 and 36A

Figure 32 is a flow chart of the “friend list registration and updating processing steps of a communication game device of the present invention.

device of the present invention.

Figures 36A,36B,36C and 36D are portions of a correspondence table showing the initial communication data in the communication game device of the present invention.

Slide: Fig 35A

There’s even an example of how the data is stored in an EEPROM! Let this be a lesson to always do your OSINT, it could save you trying to dump information off of a mystery chip encased in epoxy.

Slide: Endian-ness

One thing that the patent doesn’t address (or that I missed when reading it) is which direction the bits are stored in. You may have encountered the idea of “Big Endian” and “Little Endian” before, those words come from Gulliver’s travels and were used to describe whether people ate eggs Big-End first or Little-End first. It’s the same with numbers, does the computer eat the number’s most significant (big) end first? Or does it eat the little (least significant) end first?

Fortunately for us, the data is stored in big-endian regular old 8-bit aligned bytes. Hallelujah. I was able to interpret the bits in the same order as they were recorded in.

Slide: Message format

The patent is slightly different from the actual 2024 format. The name is in a slightly different place, and some other parts are moved around or missing. There’s still some that I haven’t totally figured out. The patent was an extraordinarily helpful starting place, though.

The checksum is right at the end though, where it says it should be. I was sweating reading about how to guess CRC polynomials, but it's just a sum mod 256. Thank goodness.

That picture is a screenshot of the web app I made to help figure all of this out, I called it the Tamagometer. It has tools to record two tamagotchis interacting with each other, record just one Tamagotchi by collecting alternating messages to bootstrap a recording, and also to edit and replay the recorded conversations to see what changing different bits does.

It's hosted on Github pages, it uses Web Serial to talk to the IR bridge (whether that's a Flipper or a DIY thing, or your DC32 badge), and I used Vue 3 for the UI.
Slide: add the emojis
I chose all of those things what I believe are good reasons, but I want to acknowledge that they do have downsides.

Github pages is free, but very much free as in beer and not free as in speech. Microsoft owns it. I chose to use it primarily so the site will stay up even if I get bored of Tamagotchis one day, because I've had my heart broken too many times by cool looking projects from a year or two ago that are just a broken link. I also like the "nothing up my sleeve" aspect of building it and hosting it directly out of the Github repository. If someone wants to fork it, or if I want to move off of Github someday, it's ready to go.

I brought in Vue because there was a lot going on, and it promised to help manage it. It's not the most bloated and disastrous front end framework, but using one at all definitely adds complexity.

I used Web serial because it’s so much easier for someone to go to a website than in it to get them to download a program and god forbid, set up a Python virtual environment, and that would be the alternative. It also let me use the web browser as my GUI toolkit, which blows everything else out of the water. Both for ease of development and for hopes of actually looking decent. There’s a reason electron apps are a thing.

The downside of Web Serial is that it only works in Chrome. I don’t like that, but it seemed better than working in zero browsers.

I also think at this point a push to add Web Serial to Firefox might actually get somewhere, since the last time it got shot down it was because Mozilla considered the API harmful. They quietly changed their opinion from “Harmful” to “Neutral” in May of 2024, so maybe there’s hope? I haven’t asked myself because I haven’t had the mental space to get flamed on Bugzilla.
https://mozilla.github.io/standards-positions/
https://github.com/mozilla/standards-positions/issues/336

Slide: Discoveries
Through a combination of looking really hard at it, trying some experiments, and massaging the bits around, I figured out how to do some things!

Slide: Infinite Money
There’s a game where your tamagotchis fight, and the winner doubles whatever money they wagered. Normally the tamagotchi randomly chooses 10, 20, or 30 points to wager, but in the tamagometer you can set it as high as 255.

I’m assuming that the tamagotchis also normally do some negotiation for the amount of money being wagered, but I’m just cutting that part of the process out and editing the message that sends the calculated final value.

510 points goes a long way, especially when you don’t actually need to buy items. Because...

Slide: Free items

You can get any item as a gift for free! Take a gift recording and edit the item byte, and you can get anything you want.

Slide: Secret items

In addition to the regular items you can get in the shop or by entering a password, some items that I’ve found no reference to on the wiki. They’re mostly Japanese food, and also a castle. And fried chicken, for some reason.

I googled these to try and figure out if someone else had found them before, and the answer seems to be no, but I did find that KFC made a Tamaogtchi-like toy, and that there was a DS game where you can make Takoyaki with Tamagotchis.

Slide: Yes you can trans a tamagotchi!

Inquiring minds wanted to know: can you trans a tamagotchi? The answer is yes, the gender is stored in the 8^{th} bit of the 11^{th} byte, the one right after the name. 0 is boy, 1 is girl. This unfortunately means that tamagotchi gender is literally binary, but you can change it at will. Gender is also stored totally separately from the character’s appearance, which is kind of interesting.