Trading Card Identity using NFCs

Derrick Hudson

5/1/2022

Abstract

NFC technology provides the fastest way to communicate with two devices within a fraction of second and is wireless. Using near-field communication (NFC) tags, the wireless functionality is utilized by storing the virtual identity of a Yu-Gi-Oh! trading card connected to the cardboard card in physical space. This is achieved by attaching the thin and flexible sticker to a card sleeve where the card is inserted. The virtual identity is recorded to the NFC chip using the ndef protocol in the form of a URI link to a digital image of the card in an online database. Any information beyond a card's image can be stored and tracked as well. Computers with the Windows operating system innately have the capacity to detect NFC tags, if connected with an NFC reader, but do not by default have the capability to execute records stored on NFCs. My project reads and executes the record stored on the NFC cards in Windows for the intended use case of streaming and recording videos.

Table of Contents

List of Figures and List of Tables

Figure 1 - relatively empty table still difficult to discern any card details

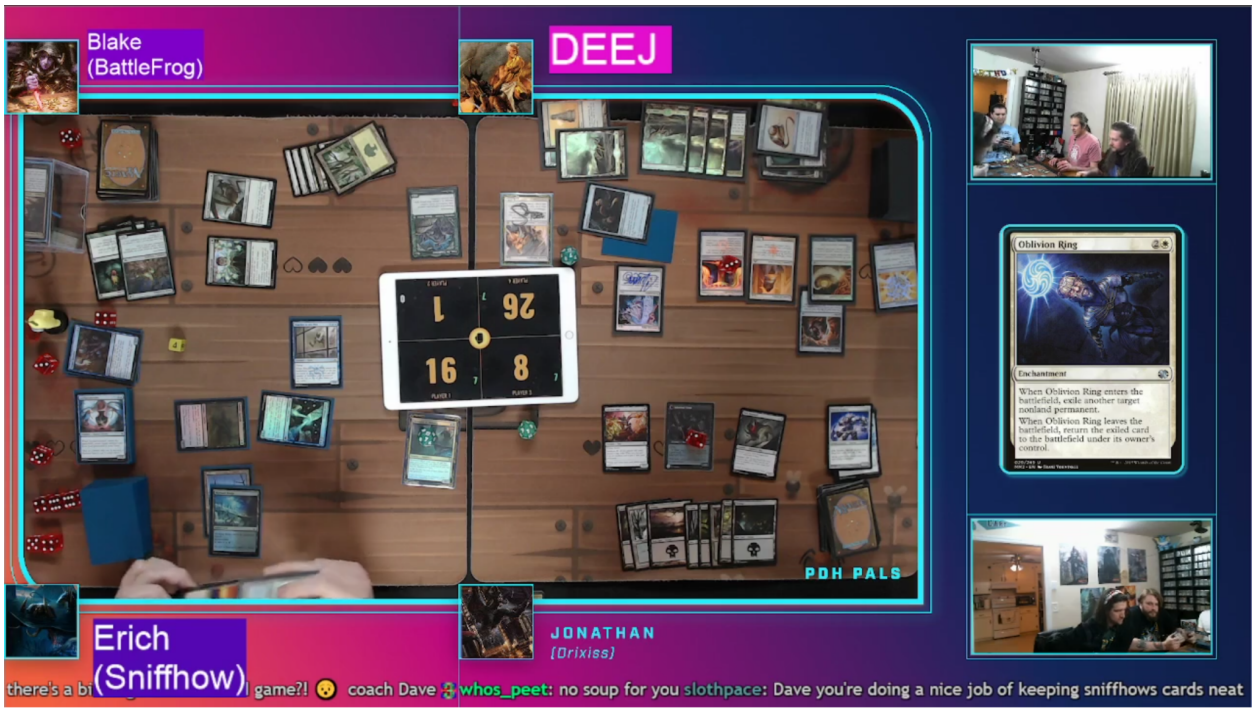

Figure 2 - messier table comparatively (typically even more cluttered) still difficult to discern any card details

Introduction and Background

My friends and I built a community around the trading card game Magic the Gathering called the PDH Pals. The tabletop gaming community is a minority subgroup of the gaming community as a whole. Of that subgroup fewer still play card games, even fewer play specifically Magic the Gathering, and in the Magic community there is an alternative way of playing that is multiplayer games of typically four players called Commander. My friends fit in a very small niche called Pauper Commander where we only play with cheap cards rarely used by anyone. Not only do we play these obscure cards in this obscure way but we also live stream the games we play and have done so twice a week for nearly eight years! Why is any of this of any relevance? A problem is posed in this streaming practice that I have chosen to address with this project. Not only would people in the broad group of gaming be confused if they happened upon the PDH pals stream, but so would the average Magic player or even Commander enthusiast. I will explain what exactly is so confusing, what we've been doing these eight long years, and what I've created to solve or help alleviate this problem moving forward.

Magic as a spectator game has its challenges. Typically any recording of a live game would be at a professional level akin to watching professional sports with accompanying budgets for commentary and visualizations to help viewers understand what they are witnessing. At times it can be a complex game and to the uninitiated it looks like a pile of cards on a table. As can be seen in figures 1 & 2 on the PDH Pals stream overlay we have a window on the right between myself and my friends where a card from an online card database shows in greater detail, and always in the proper orientation, what card has been most recently played. Jonathan, typically the most attentive of the group, manually types the name of every single card played during our four hour play sessions. The flow of our games changes because of this as we verbally declare the name of the cards we play and wait for Jonathan to type in the name of the card. Not only is this extremely tedious for whoever is typing, it's marginally tedious for everyone playing, and continuously breaks the flow of gameplay or conversation for the viewers.

I believe there must be a better way. The desired outcome of having every card that is shown in the jumbled mess that is the table of cards seen in figure 2 displayed more legibly on screen should not have the cost of distraction and labor that is our current solution of manually typing card names for hours on end. Not only would a prospective better solution be useful to the stream for Magic that I play with my friends, but any Magic the Gathering content creator. It could lower the barrier of entry for anyone who otherwise would love to play multiplayer card games or interact with a digital space like playing games remotely, recording, or streaming.

My initial solution to this problem was going to be the use of computer vision. Image detection through computer vision is a powerful tool that has sometimes troubling applications in facial recognition or the fascinatingly future-tech application of a self-driving car infrastructure. There are some hurdles that come with executing a solution to this problem using computer vision. Major points are resolution, incorrect results, and discerning images from an ever

growing mess of cards on the table. Any one of these points could be a problem for this project in particular but for my use case, all three were problems with this solution.

Using computer vision to detect a card image on the screen and return its database entry would be completely viable if a high resolution image were used. We use a 1080p webcam mounted to the ceiling. While I'm quite content with the resolution quality of our stream, it could be improved with a 4k capable camera and a 4k resolution feed. This solution has an opportunity cost of being financially costly, but it also has an obvious problem that if the resolution of the cards were greater to a certain threshold, there would be no point in having the accompanying image on the side for clarity. Given how busy the table gets no amount of resolution truly solves the issue with computer vision as an option regardless of the camera upgrade we do not plan on making any time soon. With a lower resolution image, image detection software would have to primarily look at color distinctions and the art on the cards itself. Color distinctions would not be a solution in the slightest with Magic cards because all cards are one of five colors so all you could do is reduce the massive 20,000+ card pool by 1/5th for any individual card. Card arts are all distinct but only make up roughly ⅓ of the card's physical space. Again another resolution problem with how zoomed out our camera's perspective has to be to include all information on the table.

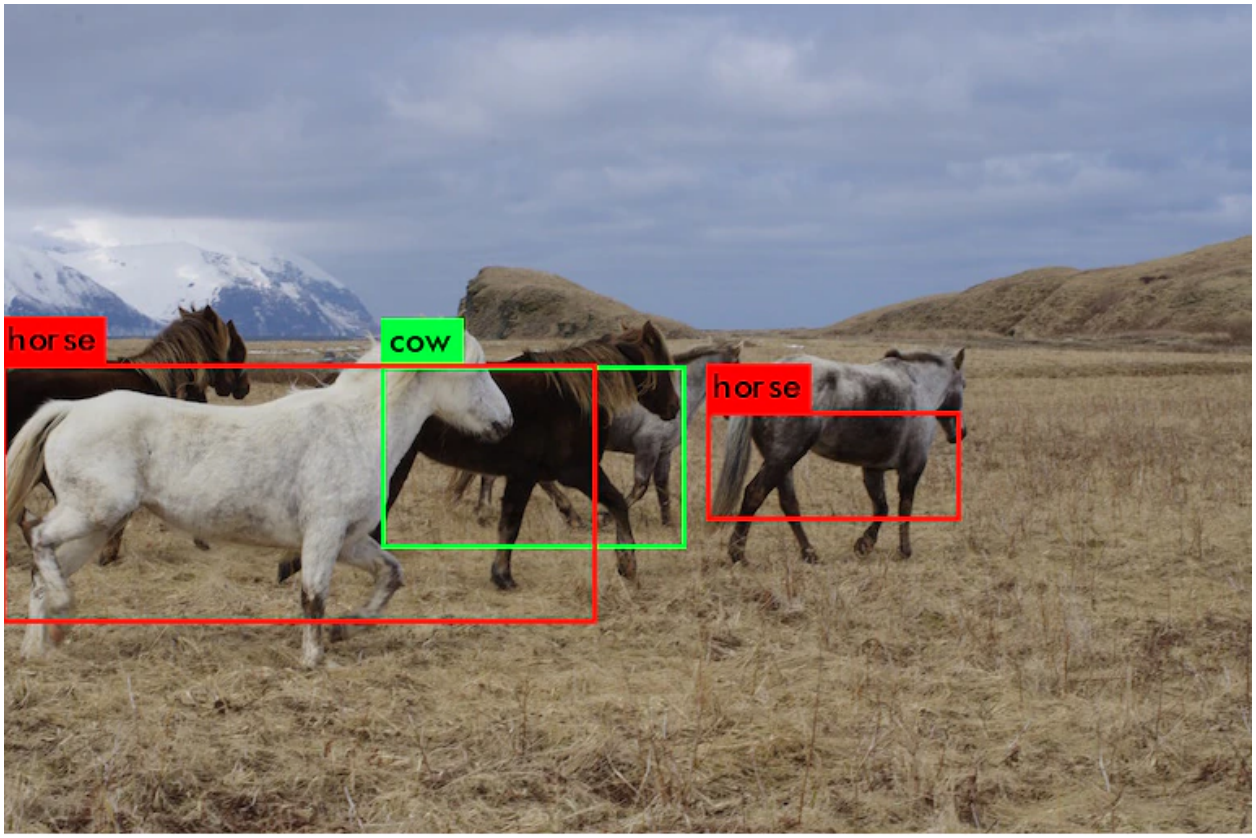

Figure 3 - Yolo 9000 struggles with partial detection and accuracy, though it is very fast.

The most troubling issue with image detection for the PDH Pals stream is how it fails in even the most sophisticated applications when images are partially visible. The YOLO 9000 (you only look once seen in Fig 3) technique for real time object detection has been a groundbreaking method that brings much hope for self-driving cars, but for the same reason the streets aren't currently dominated by self-driving cars, this method fails for card detection. YOLO 9000 would be able to quickly in real time have the capability to determine that all the cards on the table are cards, but be quite unable to determine which each card is (of the 20,000+ cardpool) with any level of certainty. Certainty can be 99% or greater when only one card image is presented with perfect lighting, no glare, and at high resolution. Without those conditions met, false positives would be frequent.

Image detection as a solution to my use case has another problem which I believe could be solved. Currently there is software that uses computer vision to build a deck list or collection table where all you have to do is load up the software on your phone and scan cards one at a time where they are added to your collection. This application is viable and quite similar to the solution I ultimately came up with though I believe it is inferior. As mentioned before this requires no glare and perfect lighting and the card would have to be brought into a very specific zoomed in space on the table. This software used by the card buying and selling websites probably uses deep learning models of training to determine which cards are which. Since the same results can be achieved without having to retrain every time a new set of cards is released and the litany of other potential issues, I opted out of pursuing computer vision as a solution at this point.

What I was looking for was a solution that was discrete. I wanted something that my friends wouldn't have to think to use and have no chance of returning the incorrect results. If we had a small portion of our game space dedicated to an up close camera then we'd have to reach into that space for every card played to scan in the card's image and it could still fail to return the proper image or anything at all. That's when NFC chips come into the picture.

Figure 4 - NFC tag/sticker

Design, Development and Test

Most of this project is the result of piecing together many different parts, physical and digital. There are the NFC stickers,the NFC reader, and the NDEF protocol [1] necessary for the two to be able to communicate. I had to use a third party driver for the reader to work with my library of choice nfcpy [4]. There is the api for the card image database which is what was stored on the physical NFC tags. Lastly the nfcpy library has a few dependencies, winUSB, libusb, and docopt, to be able to function. The logic of how I sequenced piecing these parts together will be described in this section.

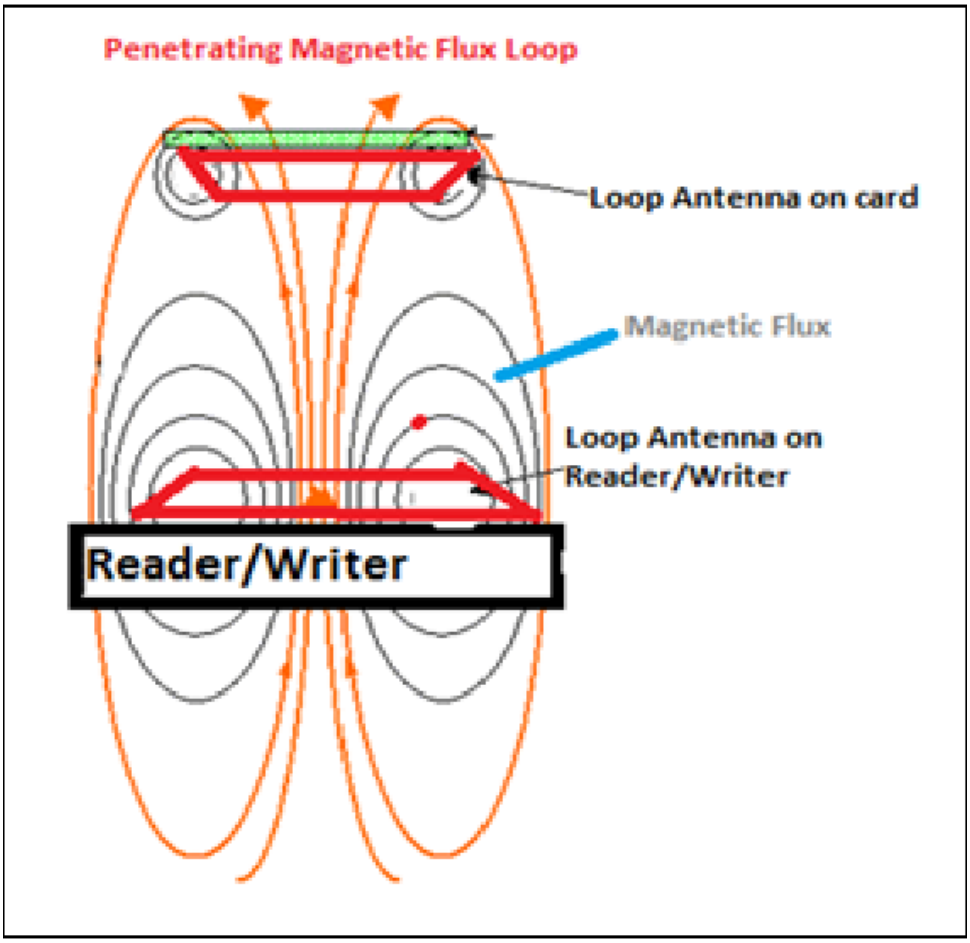

Figure 5 - How NFC's work

What are NFC's? NFC stands for near-field communication, it's a proximity based technology where a magnetic field is generated from an NFC reader which powers the NFC chip. An NFC chip is RFID technology (radio frequency identification) which has historically been used in retail for loss prevention. Where older RFID tags would trigger in a range of the doorway of a shop or the gate at a toll, NFCs are much shorter range. Think inches for NFC and feet for RFID. The advantage is typically security but for my application the shorter range allows for precision. NFC chips are also superior to RFID tags because they have both read

and write capability. A few notable features of the NFC tags that made it a great fit. They are extremely thin, flat, and small (seen in figure 4). They also don't have a battery which allows them to be so thin, because they are powered by the NFC reader when they get in proximity (see figure 5). While the amount of data they can hold is quite limited, in the scale of bytes, it's sufficient for holding a card's virtual identity.

NFC tags are the perfect solution to this problem. Everyone playing would still have to reach into a designated space on the table to register the card but instead of it being a small space with a camera that could produce inaccurate results, there would be an NFC reader where you tap your card for less than a second without regard for glare, angle, or worrying if your thumb was in the way. From this point onward the project pivots away from the original use case of the stream I have with my friends but rather a different card game with all the same issues, Yu-Gi-Oh! This decision was made because all of my Magic cards are in the PDH Pals studio in Cary, NC while all of my Yu-Gi-Oh! cards are with me in Boone. I have a Youtube channel where I upload Yu-Gi-Oh! content so the use case still holds as well as all steps involved are the same for both card games. The only thing that changes is the database used for card images.

Figure 6 - Yu-Gi-Oh! card's id number circled in blue

For Magic The Gathering the online card database we use is Scryfall. Every time someone plays a new card we verbally announce which card is played and if Jon is paying attention then he'll type that card name into Scryfall's search window to pull up an image of the card as seen in our stream's overlay. For Yu-Gi-Oh! I'll be using a similar card database called Ygoprodeck. You can search for a card by its name exactly like Scryfall or use an alternate method that I utilize to conserve memory which is important considering I am working in the scope of bytes where word length matters. Each Yu-Gi-Oh! card has an eight digit unique identifying number located in the bottom left of the card (figure 6). This number has significance

because while the example of Des Frog is eight characters just like its id number, some cards have comically long names like “Exodius the Ultimate Forbidden Lord” clocking in at thirty five characters. Ygoprodeck has an api that makes the same observation and always categorizes cards by the id number. Using this database’s api for virtual card images I now have images to store on the NFC tags in the form of a url.

Figure 7 - NFC sticker inside of sleeve on the left, none in the right

Cards in either of these games are considered collectable. To preserve the condition of these collectables almost all of the community uses sleeves, plastic covers that protect the cards. I will be attaching my NFC stickers to the inside of the sleeves (figure 7). I’m sure the stickers wouldn’t harm the cards if they were attached directly, but I’d rather take the risk with a $4 pack of sleeves than on the backs of cards that could be worth thousands of dollars. As expected the stickers are so thin that they are hardly distinguishable and shuffle just as well as any other sleeved deck of cards.

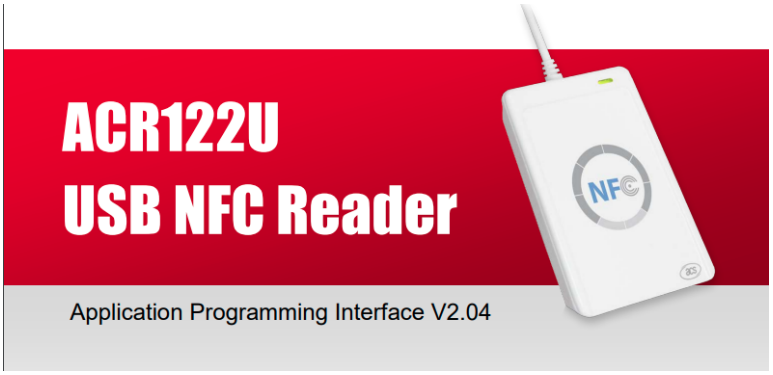

Figure 8 - ACR122U NFC Reader

I now have NFC tag stickers that are attached to sleeves with Yu-Gi-Oh! cards inside of them and a strategy for loading in corresponding card images using Ygoprodeck's database api. It is possible to use the NFC reader on my iphone to write the card image to the NFCs or to use a usb NFC reader and write them using my computer. This project's goal is to use a pc to read the cards and it's faster to write an entire deck with this method rather than using my phone so this is where I had to make the important decision as to which NFC reader I need to purchase. I wasn't sure which NFC library I was going to use but I had settled on two, ndefnfc [2] which is a C# library or nfcpy written in python. A little Amazon research directed me to the ACR122U NFC reader which was more affordable and seemingly as robust as the more costly options available. According to the documentation for both of these libraries this particular brand of NFC reader was fully supported so this was the route I took (figure 8).

Figure 9 - Reader detects Des Frog but not Exodius because of presence of NFC tag

This reader has its own driver which is perfectly functional with Windows OS and the software I used to write the NFC tags called nfctools [3]. A major roadblock was discovered at this stage. Windows has vulnerabilities for NFC readers with the primary intention of using NFC tags for login information for schools and business purposes. This is mostly a relic of the past though some businesses may utilize this functionality. The relevance of this is that there is an operation that is always running by default called the "Windows certificate propagation service" which gets in the way of reading or writing data to NFC's. Disabling it is quite simple with administrator permissions, perfectly fine for my personal computer but proved to be a pain with the computers on campus. Once the service is disabled the reader works as it did before but Windows doesn't take over and interrupt like it did before, excellent! Using nfctools writing the url for each card was a breeze.

```
Addr. 00 : UID0 - UID2 / BCC0 >> 04:d1:38:65
Addr. 01 : UID3 - UDI6 >> 12:53:70:80
Addr. 02 : BCC1 / INT. / LOCK0 - LOCK1 >> b1:48:00:00
Addr. 03 : OTP0 - OTP3 >> e1:10:3e:00
Addr. 04 : DATA >> 03:3c:d1:01
Addr. 05 : DATA >> 38:55:04:73
Addr. 06 : DATA >> 74:6f:72:61
Addr. 07 : DATA >> 67:65:2e:67
Addr. 08 : DATA >> 6f:6f:67:6c
Addr. 09 : DATA >> 65:61:70:69
Addr. 0A : DATA >> 73:2e:63:6f
Addr. 0B : DATA >> 6d:2f:79:67
Addr. 0C : DATA >> 6f:70:72:6f
Addr. 0D : DATA >> 64:65:63:6b
Addr. 0E : DATA >> 2e:63:6f:6d
Addr. 0F : DATA >> 2f:70:69:63
Addr. 10 : DATA >> 73:2f:38:34
Addr. 11 : DATA >> 34:35:31:38
Addr. 12 : DATA >> 30:34:2e:6a
Addr. 13 : DATA >> 70:67:fe:00
Addr. 14 : DATA >> 00:00:00:00
Addr. 15 : DATA >> 00:00:00:00
Addr. 16 : DATA >> 00:00:00:00
Addr. 17 : DATA >> 00:00:00:00
Addr. 18 : DATA >> 00:00:00:00
```

Figure 10 - Example dump file of the contents of an NFC's NDEF format

There is a bit of a technical point of understanding at this stage. The NDEF protocol is the structure used for formatting the data on NFC tags in such a way that the reader can understand. Most of the protocol's structure is allocated to the payload or the contents of the stored data, but the first handful of bytes are for things like the tag's id number (different from the Yu-Gi-Oh! card unique id number), whether or not the tag is locked (which when locked can not be reversed), and most practically there's a header byte. The header byte is used to save space on the tag. For this application I use the header URI identifier code 0x04 which represents "https://www.". The rest of the data on the NFC is the payload which is the url from the database api, minus the contents of the identifier code(figure 10).

Results

Now all that's left is the application! After much deliberation I decided to use the nfcpy library and settle on Python. The way this library works is it takes each of the Window's NFC vulnerabilities and creates wrapper functions that can be called in Python. It also easily parses the NDEF protocol to painlessly read the contents of the NFC payloads. I read through the documentation and it seemed simple enough to read the contents of an NFC but even when using the examples provided nothing was happening. In the comments of the part of the library where the specific readers are called, my ACR122U had a side note of "due to the proprietary nature of drivers, both reading and writing are not currently supported." It wasn't until I was already completely committed to this library and my particular NFC reader did I learn that the scope of my project had become only storing an image of a card rather than utilizing NFC's

ability to both read and write to have at the very least a tracker of how many times the card had been scanned to track use. This does still fulfill the requirements of my intended use case of assisting with the flow of our streams and videos, but it would have been nice for this nugget of information to be in the library's documentation so I could have purchased any of the other 11 supported NFC readers.

```
** waiting for a tag **
Type2Tag 'NXP NTAG215' ID=04D01012537080
NDEF Capabilities:
  readable  = yes
  writeable = yes
  capacity  = 492 byte
  message   = 60 byte
NDEF Message:
record 1
  type = 'urn:nfc:wkt:U'
  name = ''
  data = b'\x04storage.googleapis.com/ygoprodeck.com/pics/31786629.jpg'
```

Figure 11 - Command line reading of an NFC tag using nfpy library

After the major hiccup I learned that the proprietary driver which worked perfectly for writing the contents of the NFC tags was the problem and that there was an alternative available. Using a nifty tool Zading [5] I was able to automatically find a third party driver by detecting my reader which nfcpy supports but has the bottleneck of only reading NFC's. The logic of the code I had written before the big news was functional and could easily read the contents of a tag (figure 11). This was very simple to test because of the discrete nature of this step. It either scanned or didn't, after the driver update it always scanned the contents of the NFC.

The only thing remaining is to automatically launch the URI payload of the scanned NFC. A conditional statement checking for the header's URI identifier code of 0x04 and a method that takes that data and returns it launching it in my computer's default browser of Chrome is all that was necessary. There was a small hitch of having to toggle on and off the "nfc frontend" which is just a listener for the reader. There is a flag in the library for whether or not there is currently an NFC within the reader's field so the only thing that needs to be done there is to toggle off and on the frontend so the next card can be read in. Again the discrete nature of the outcome makes testing trivial because the cards either scan or don't and after hundreds of successful scans I concluded that this method works.

The objective was originally to have a way to bridge the gap between the physical world and the virtual in the domain of card games. Although I would have liked to push the constraints of NFC's to this aim, ultimately the goal has been met. Where before the monotonous task of manually typing in card names during stream into a card database disrupted the flow of gameplay, now the same result can be achieved gracefully with a simple tap and everyone can focus on playing without sacrificing the viewing experience. While my application is running,

cards with a corresponding URI payload to a card image database will be loaded in Chrome (or any default browser) when tapped into an ACR122U NFC reader on Windows OS.

NFC technology does perfectly fit my intention for this project. The utilization of both the read and write capability was part of the initial goal of the project but with the hardware limitation of the specific reader I chose I had to focus on the much more important aspect of displaying a card image rather than gathering superfluous data for a litany of uses like analyzing data or giving feedback in real time based on a dynamic “deck state”. This could have been overcome by replacing the hardware which was out of my budget, working with the other C# library which is beyond my understanding, or pivoting the project entirely into writing a driver for the ACR122U NFC reader for nfcpy which is well beyond my time scope.

The other major problem I encountered was earlier on with the Windows certificate propagation service. It was difficult initially to understand why the third party NFC writer was doing nothing, but it didn’t take much research to uncover the problem with any NFC device on Windows. There were multiple resources that pointed to this as the problem and immediately after disabling the service everything worked as it should and I never looked back.

Conclusion

The amount of time I spent on the computer vision approach to solving this problem had an effect on my progress. I’m glad I took the necessary amount of time to change my approach to something I thought would be simpler. Though the NFC approach did prove to have many more moving parts than I anticipated. Ultimately I’m very happy with the result of the NFC approach and look forward to using this for the years to come of gaming with my friends.

The goal of having a streamlined method of quickly scanning in cards for display in a stream setting has been met. This project was much more complicated than I initially anticipated. While I know the nuances of some of the things I wanted to implement are possible, the intended use of NFCs were the only resources I was able to find and learn from ultimately making any deviation from that surprisingly difficult.

I would love to use NFCs in making a hybrid board game/video game where the read/write capabilities are utilized to have a memory of past games played to be used as callbacks in future new playthroughs. The deck I use in Yu-Gi-Oh! is an overly complex combo deck that has hundreds of variations of sequences of play that I struggle to teach others how to employ. The next step I’d like for this project is to have an educational tool with a window displaying a percentage chance of “continuing to combo”. As specific cards are scanned in the deck’s state would be tracked and the mathematical odds of continuing would be calculated, if the odds hit 0% then the user definitely made a mistake and would know exactly when they made it and ideally learn to not do that next time.

References

[1] O'reilly Media. 2022. Introducing NDEF. (January 2014). Retrieved March 2, 2022 from https://www.oreilly.com/library/view/beginning-nfc/9781449324094/ch04.html

[2] andijakl. 2022. ndef-nfc. (2017). Retrieved March 2, 2022 from https://github.com/andijakl/ndef-nfc

[3] wakdev. 2022. NFC Tools - PC/Mac. (2022). Retrieved March 2, 2022 from https://www.wakdev.com/apps/nfc-tools-pc-mac.html

[4] nfcpy. 2022. nfcpy. (2022). Retrieved March 2, 2022 from https://github.com/nfcpy/nfcpy

[5] Pete Batard. 2021. Zading (2021). Retrieved March 2, 2022 from https://zadig.akeo.ie/