

Nelson Torres – Graduation Portfolio

Master's of Science in Computer Science

Upon my graduation from Middle Tennessee State University in December 2022, I began Vanderbilt's Master's of Science in Computer Science program through their Graduate School. Since starting the program in the Spring of 2023, I have earned the grades below with the courses listed:

- CS5260 – Artificial Intelligence – Spring 2023 – Ronald Hedgecock - B
- CS5279 – Software Engineering – Spring 2023 – Yu Sun - A
- CS5262 – Foundations of Machine Learning – Summer 2023 – Peng Zhang - A
- CS6388 – Model Integrated Computing – Summer 2023 - Tamas Kecskes, Janos Sztipanovits – A-
- CS8395 – Quantum Computing – Summer 2023 – Charles Easttom - A+
- CS5253 – Parallel Functional Programming – Fall 2023 – Douglas Schmidt - A
- CS5283 – Computer Networks – Fall 2023 - Anriuddha Gokhale - A-
- CS8395 – Digital Forensics – Fall 2023 – Charles Easttom - B-
- CS5288 – Web-Based System Architecture – Spring 2024 – Edward Klein - A
- CS8395 – Microservices – Spring 2024 – Douglas Schmidt - A

My professional aspirations are to continuously grow as a software developer and engineer at companies that are creating new technologies. My personal interests are always geared towards programming and debugging rather than to someday transition into a managerial position. I enjoy being able to create and innovate, rather than delegate. I enjoy problem-solving and being able to work with new technologies, so I know my professional end goal is to work with companies that are constantly evolving and pioneering new technologies as a Software Developer or Engineer.

I have earned a GPA of 3.74 and a total of 30 credit hours. Throughout my time, I have learned about dozens of topic areas in Computer Science and personally feel most interested in Artificial Intelligence / Machine Learning topics, as well as Large Language Models, Big Data Analytics and Parallel Programming. I hope to one day be able to pursue a Doctoral Degree in either computer science or computer engineering.

Curriculum Vitae

Upon graduating from high school in 2018, I began my academic journey at the University of Tennessee in Knoxville (UTK). I transferred to Middle Tennessee State University (MTSU) for financial reasons the start of my sophomore year in 2019. I retained my Computer Engineering major from UTK and tacked on a secondary major in Applied Physics, with a concentration in Computer Science and a minor in French. In the March of 2020, I completed a Data Analytics Bootcamp at Vanderbilt University where I truly began delving into the technologies used in the industry rather than the concepts thought throughout my undergraduate degree. I graduated with these majors in December of 2022 from MTSU with a 3.4 GPA and earned a Silver Scholar Research Award for my work and thesis with Dr. Erenso on Measuring radiation dosage of BT20 cancer cells. I began full-time employment as a Full-Stack Software Developer at IPro Systems in Hendersonville, TN during my last semester at MTSU although prior to this I did work as a residential contractor, database manager, controls engineer intern and physics tutor. I also started my Masters in Computer Science at Vanderbilt University during the Spring of 2023 with an expected graduation of May 2024.

Through all my academic and professional experience, my expertise has shifted from time to time and will most likely continue to change as technologies do but, at the moment, my most commonly utilized languages are C++ (used throughout my undergraduate degree), Python (used for my professional career), and Java (used throughout my graduate degree). While these are my most commonly used languages however, they are far from the only languages and skills I have honed in my time working and studying. I work with Node.js to connect with the API that I develop in Python at work. I leverage AWS cloud computing and Apache Airflow to initiate and automate Spark ETL tasks in both Python and Java if the application calls for it. I have a handful of personal projects that use brand new technologies I was never introduced to throughout my academic or professional career such as C++ for Arduino, React, Tkinter etc. Along with all this, one of my greatest strengths is my deep understanding of foundational computer science concepts making it so I am able to learn and adapt to any new framework or tech stack I need to work in as I understand how it all works under the hood.

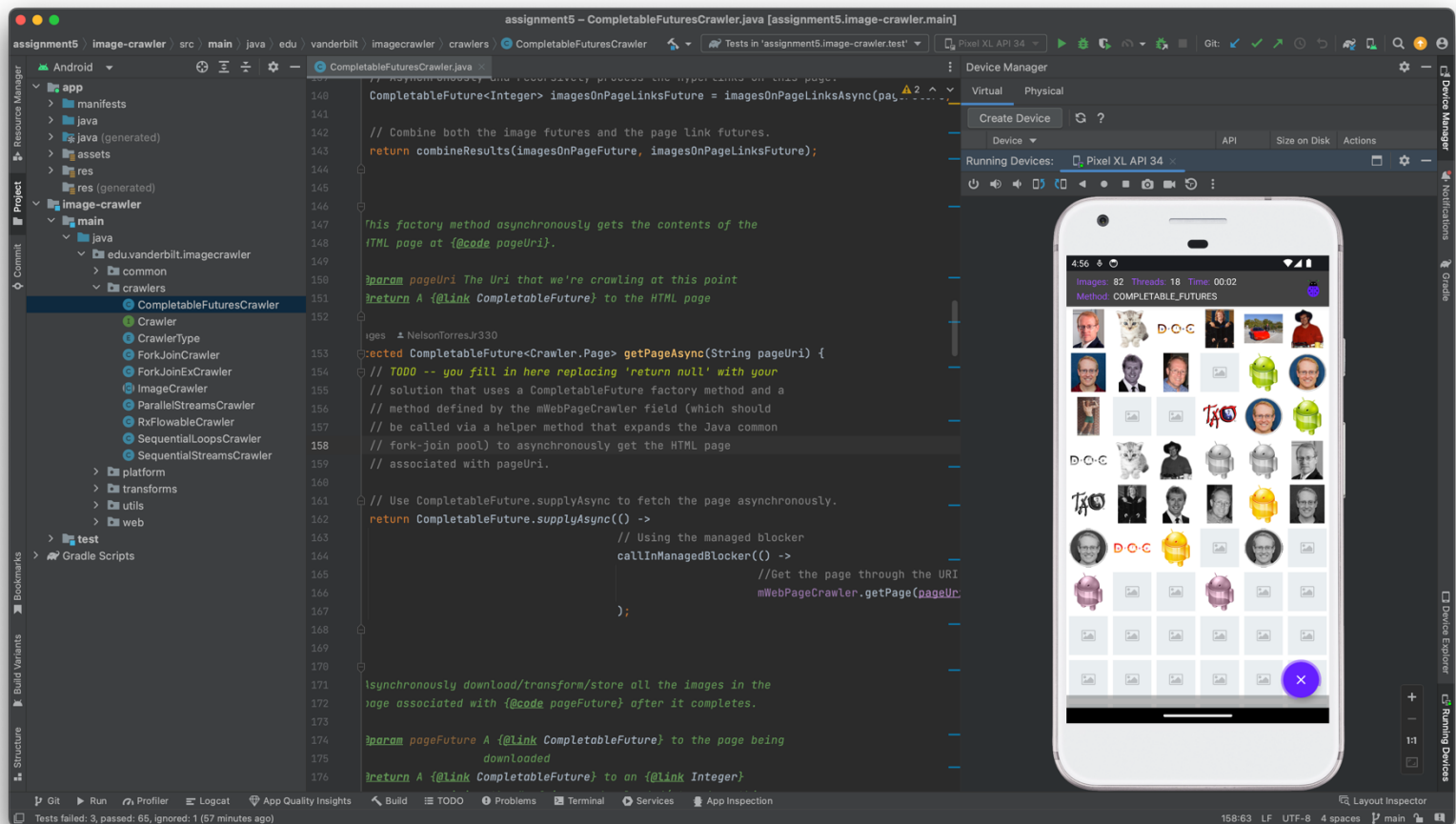
Knowledge and Mastery of Parallel Functional Programming

I completed Vanderbilt's CS5253 course in Parallel Functional Programming in the Fall of 2023 with Dr.Schmidt. The course's objectives were to teach students how to develop "reusable parallel programs [that] can be designed, implemented, optimized, validated, maintained and enhanced by applying effective functional and object-oriented development practices, patterns, and frameworks". During my time in the class, the assignments were all built on top of the same project so the final product is a modular and polished result.

The project is an image downloader application with a GUI created on Android Studio. Most of the source code that built the application and GUI was provided by Dr.Schmidt, leaving the student to just handle implementing new kinds of downloading methods, entirely in Java. The 7 methods implemented in the final version of the application are Sequential Loops, Sequential Streams, Parallel Streams, Completable Futures, RX Flowable, Fork Join, and Fork Join Ex (Sequential Streams in conjunction with Fork Join). For the sake of this portfolio, I will only dive into explicit detail on the Completable Futures methodology, however, if interested, the full code with every implementation is publicly available at <https://gitlab.com/Nelsontorresjr330/cs-5253-fall-2023> .

First released in 2014 with Java 8, Completable Futures piggybacks off of Java's already-in-place Future class. These future classes in Java are references to the results of asynchronous calculations within the project and Completable Futures were designed to handle the limitations of the generic future class. In a nutshell, Completable Futures are Java classes used for asynchronous parallel programming that allow the developer to create non-blocking tasks on separate threads rather than on the main thread. This allows for the main thread and the additional threads to run in parallel and increase performance.

As demonstrated in the screenshot of Android Studio below in Figure 1, the application displayed on the right is running the Completable Futures Method and is loading in numerous pictures while others have not been rendered entirely yet. This is due to the asynchronous nature



of completable futures. To

Figure 1: Completable Futures
initialize a completable future in simple applications, the syntax is simple enough:

```
//Create a new CompletableFuture Instance.
CompletableFuture<String> completableFuture = new CompletableFuture<>();
```

However, in this application, we were tasked with creating CompletableFutures dynamically per image that needed to be downloaded and later transformed. To do this, we also needed to utilize Java sequential streams to check if the current depth is within the max allowed depth and to see if the URI had been visited before, as the images that were to be downloaded came in through a list of URIs that were not necessarily distinct. Below is the code snippet that did this, note the function is called as many times as there are URIs sent into the main application.

```
//Create a stream
return Stream
    //Of all the page URLs
```

```

.of(pageUri)
// Filter out the URI if the depth is > the MaxDepth
.filter(url -> depth <= mMaxDepth
    // Also filter it out if the URI has been visited before
    && mUniqueUris.add(url))
// Apply getPageAsync to all URLs that pass
.map(this::getPageAsync)
//Map the pageFuture
.map(pageFuture ->
    //Through the getter for the Page
    imagesOnPageAndPageLinksAsync(pageFuture, depth))
// Get the first CompletableFuture<Integer>
.findFirst()
// Or a future with zero if the stream is empty.
.orElse(mZero);

```

In this snippet, the function is passed in a string containing a pageUri and a depth that the task is being executed on. We initialize the stream with the pageUri and filter out the Uri if it is not unique or if it exceeds the max depth. Then the page goes through and calls a helper function on the filtered pageUri called getPageAsync, shown below.

```

// Use CompletableFuture.supplyAsync to fetch the page asynchronously.
return CompletableFuture.supplyAsync(() ->
    // Using the managed blocker
    callInManagedBlocker(() ->
        //Get the page
        through the URI
        mWebPageCrawler.getPage(pageUri))
    );

```

This helper method is what creates the CompletableFuture object by utilizing the supplyAsync method within the class. The supplyAsync method will take in a supplier and begin its execution on a separate thread without blocking the main thread. In this case, we are working with pulling in images which adds a slight bit of complexity in handling the data so a managed blocker is necessary, but for many applications, it isn't. The supplyAsync method is essentially the value-returning version of runAsync as they both execute tasks on separate threads and return CompletableFuture Objects.

In a simpler example, we can get back the results of a CompletableFuture by using the get or getNow method to pull in the results of a CompletableFuture, with the only difference between the two being that getNow provides a default result if the task had not finished its execution.

```

//Pull the result of the completable future and print it,
//If it's not finished, print Hello World
System.out.println(completableFuture.getNow("Hello World"));

```

However, in the case of our program, the results of our completable future are pulled in through the thenApplyAsync method since we need our tasks to automatically finish when the future completes rather than waiting to be grabbed by the program at the end of execution and essentially blocked. The difference between thenApply and thenApplyAsync is that the

thenApply utilizes the same thread as the original CompletableFuture object whereas thenApplyAsync executes in a separate thread derived from the ForkJoinPool if no other executor is specified.

```
//On the page future passed in
return pageFuture
    // Asynchronously apply
    .thenApplyAsync(
        //The getter for the images on this stream
        this::getImagesOnPageStream)
    // Afterwards, Asynchronously compose
    .thenComposeAsync(
        // the images through the processImagesAsync method
        this::processImagesAsync);
}
```

In our example, this function is passed in a pageFuture provided by the getPageAsync code above and uses thenApplyAsync to get the images on the page. Once thenApplyAsync finishes pulling in the images, thenComposeAsync will execute the processImagesAsync method. The thenComposeAsync method is similar to thenApplyAsync but it instead returns another Completable Future rather than the results of the method run as thenApplyAsync does. The purpose for this is unique to this specific project as after the first image is pulled in and processed, the program transforms them and performs other operations on them to further emphasize the usages of Java streams and how to properly use Completable Futures.

One more part of the project I would like to highlight is after all the methods for different types of parallel and sequential processing have been implemented, we can see and benchmark them side by side. Below is a table of the time results I obtained with my laptop and the methods used for each.

Sequential Loops – 1 Thread	6 Seconds
Sequential Streams – 1 Thread	6 Seconds
Parallel Streams – 6 Threads	5 Seconds
Completable Futures – 26 Threads	4 Seconds
RX Flowable – 1 Thread	6 Seconds
Fork Join – 28 Threads	4 Seconds
Fork Join Ex – 28 Threads	4 Seconds

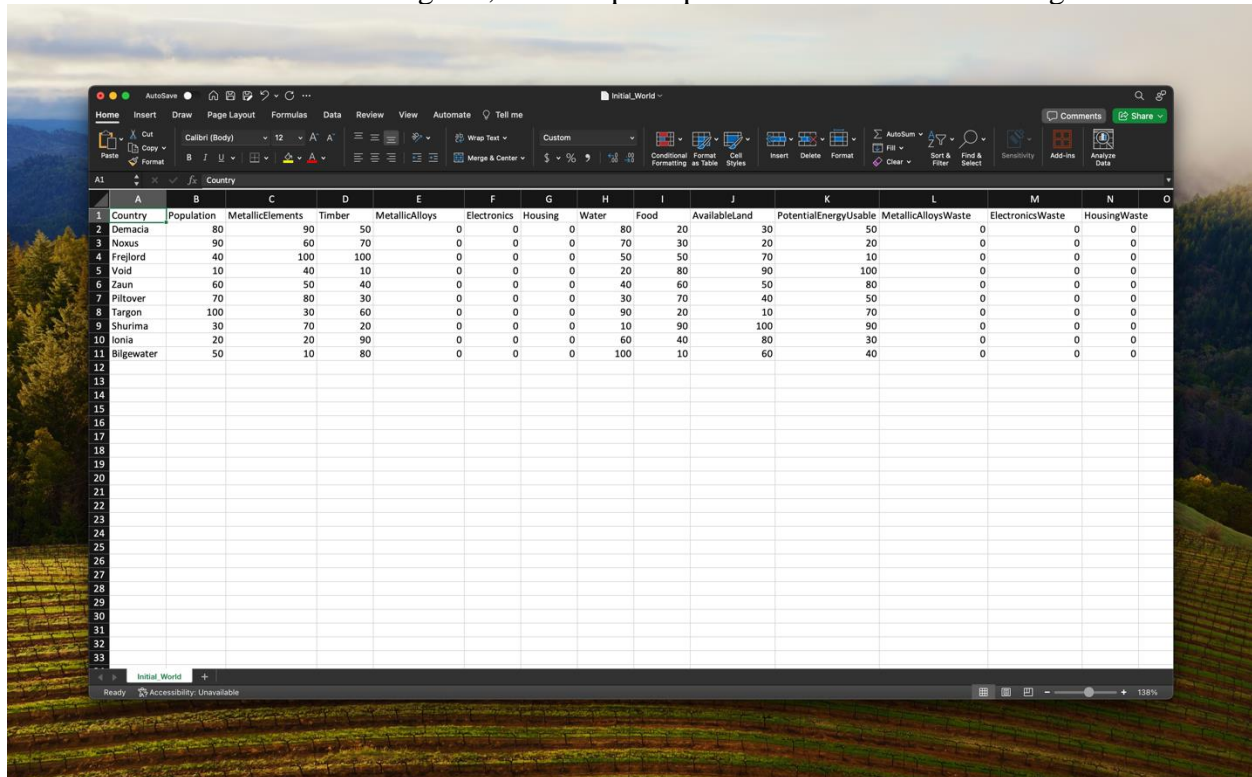
While the difference in seconds may seem negligible, it's important to remember that only 96 images are ever pulled in from the source in this instance so there isn't too much computing going on in the first place. The biggest takeaway for me is understanding the nature and usages for each of these different methods, as well as being able to see their results side by side and just how much of the CPU is actually being utilized simultaneously.

Knowledge and Mastery of Artificial Intelligence

Throughout my first semester at Vanderbilt, I completed Dr.Hedgecocks Artificial Intelligence course. This course was designed to teach the “Principles and programming techniques of artificial intelligence. Strategies for searching, representation of knowledge and automatic deduction, learning, and adaptive systems”. Throughout the course, we learned numerous searching methods paradigms and techniques, along with more complex AI topics such as Bayesian Networks and Markov Models. The final project for this course was to create a civilization simulator program that will be able to process files that hold several theoretical countries’ metadata and output a list of moves that would grant the country of choice, the highest “happiness score”.

In making the civilization simulator, the students were allowed to implement their choice of search methods and implement whichever techniques they chose to optimize it. I wrote mine entirely in Python and utilized Heuristic Depth First Search in conjunction with Expected Utility Theory and Discounted Rewards. Along with this, I also added in Multiprocessing Parallelism to speed up execution times and test the limits of the program.

For more context on the game, an example input file is shared below in Figure 2.



Country	Population	MetallicElements	Timber	MetallicAlloys	Electronics	Housing	Water	Food	AvailableLand	PotentialEnergyUsable	MetallicAlloysWaste	ElectronicsWaste	HousingWaste
Demacia	80	90	50	0	0	0	80	20	30	50	0	0	0
Noxus	90	60	70	0	0	0	70	30	20	20	0	0	0
Frejlord	40	100	100	0	0	0	50	50	70	10	0	0	0
Void	10	40	10	0	0	0	20	80	90	100	0	0	0
Zaun	60	50	40	0	0	0	40	60	50	80	0	0	0
Piltover	70	80	30	0	0	0	30	70	40	50	0	0	0
Targon	100	30	60	0	0	0	90	20	10	70	0	0	0
Shurima	30	70	20	0	0	0	10	90	100	90	0	0	0
Ionia	20	20	90	0	0	0	60	40	80	30	0	0	0
Bilgewater	50	10	80	0	0	0	100	10	60	40	0	0	0

Figure 2: Example Country Resources Sheet

Once the input data is read and parsed, there is another input file that contains all the weights of each resource for all the countries similar to the table below.

Resource	Weight
Population	0.6
MetallicElements	0.4

Timber	0.4
MetallicAlloys	0.8

Except filled with all the resources noted in the countries' metadata file. Once the program has parsed all the data, its end goal is to try and output a list of the best possible moves to provide the country with the highest happiness score. This total happiness score is determined by getting the sum of all the weights of each resource multiplied by the total number of that resource for that country. For example, if a country had a population of 100 and a food value of 200 and the values for population and food were both 1, its total happiness score would be 300. Possible moves a country can make to get to the highest possible happiness is another variable we need to consider as numerous actions can be taken on all the resources.

All the possible actions are trading resources with another country (but you must consider the likeliness they will accept the trade), refining a resource (each defined in the templates within input files, also parsed), or clearing out wasteful resources. The main loop within the program is visualized below in Figure 3.

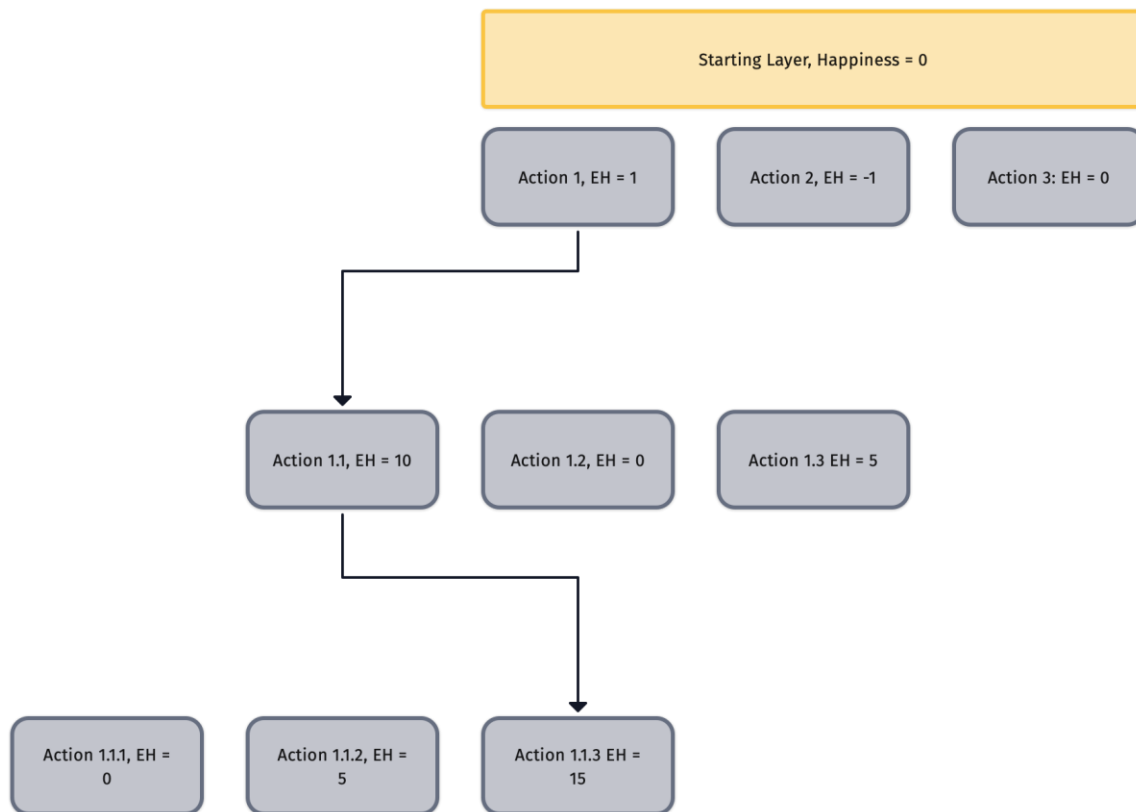


Figure 3: Heuristic Depth First Search Loop

For the sake of simplicity, there are only three actions in the visualization, when in reality there are 10s to 100s based on the number of resources and available transformations. In this visualization though, at the start, we begin with a happiness of 0, and the program calculates the expected happiness of each possible action and depending on which action has the highest Expected Happiness after completion, that action will be taken first and that path will be

followed. Within the civilization simulation, calculating expected happiness can get very hairy and difficult to fine-tune.

```
def expected_utility(country, x0 = x_0):  
    EU = schedule_success_prob(x0) * country.get_discounted_reward() + ((1 - schedule_success_prob(x0)) * C)  
    return EU
```

The code above is the highest level of all the integrated calculations needed to determine this expected happiness. After the action is decided however, the program continues to repeat this cycle of determining the highest expected utility and following that path all the way through until the max layer is reached (set as a param by the user upon execution). Once reached, the program back tracks one layer and will run the next highest expected happiness path until that path reaches its max layer or if all possible paths have been explored at that layer already, the program back tracks another layer, until all routes have been explored and the program will output a list of actions with the best routes found for that run in an output file similar to the one below.

```
Best EU: 1.39937842875328766  
['AvailableLand', 'To', 'Zaun', 1]  
['PotentialEnergyUsable', 'From', 'Piltover', 1]  
['PotentialEnergyUsable', 'From', 'Zaun', 1]
```

In this specific run, the program starts off by giving away AvailableLand as the first action, then taking in PotentialEnergy From Piltover and Zaun before ending. This first giveaway of land might seem unintuitive at first but the AI program has a built-in Logistic Function for Acceptance Probability and it was able to recognize, if it grants this land to Zaun, Zaun will be a happier country and thus more likely to grant back Energy to the simulated country later on. Part of this calculation is shown below and is just another one of the layers of complexity and context the AI model is able to account for when making these decisions.

```
#https://en.wikipedia.org/wiki/Logistic_function  
def country_accept_prob(country, x0 = x_0):  
    try:  
        exponential = ((-k)*(country.get_discounted_reward()-x0))  
        denominator = 1.0 + (math.e ** float(exponential))  
    except Exception as e:  
        print(f"Overflow Error, exponential = {exponential} \n{e}")  
        exit()  
    return (1/denominator)
```

To recap, this project utilized the AI techniques of Heuristic Depth First Search, Expected Utility Theory, Discounted Rewards, and Logistic Functions for Acceptance probability to simulate a civilization game. All these techniques and more were discussed at length throughout the Dr.Hedgecocks CS5260 Artificial Intelligence course and while I am not able to share all my assignment solutions, my version of the Civilization game source code is fully available at <https://github.com/Nelsontorresjr330/World-Trade-Sim>.

Communication Skills in Computer Science

My abilities to properly communicate in computer science fields have grown tremendously throughout my time at Vanderbilt but the course that most impacted this growth was Dr. Easttom's Digital Forensics course, CS8395. The syllabus described in this course emphasizes the use of labs and lab reports that require the "application of theory, development of forensic arguments, and production of forensic papers". Throughout the course, there were a total of 10 lab reports written, each requiring a "graduate-level understanding of operating systems, distributed systems, and networking" to complete, with an emphasis on whichever topic was being discussed that week.

Most all the labs followed the same format with a title page, an introduction to the lab, personal background and qualifications, compensation expectation, chain of custody, analysis, and conclusions. This format was designed to simulate the actual format of a genuine forensic report, hence the compensation portion. The bulk of these assignments was the analysis section, where I needed to properly demonstrate the exact steps followed and their results so the reader would be able to follow along, a screenshot from my analysis of the Android Debugging Bridge is shown below in Figure 4.

Conduct Independent Inquiry in Computer Science

Similar to my previous section, every course I have completed at Vanderbilt has increased my ability to conduct independent inquiries in computer science, whether it be through researching a new method in Java to aid in a project or watching tutorials on UMLs to be able to design a project properly. However, the course that most explicitly emphasizes individual research and inquiring is Dr. Easttom's Introduction to Quantum Computing course.

Per the syllabus description, "Fundamental concepts including quantum hardware, logical qubits, quantum algorithms, and quantum programming will be covered. While prior course work in linear algebra and physics would help a student, they are not required". As one might expect, Quantum Computing is a massive field that could never truly be completely mastered over one semester so Dr. Easttom assigns 4 in-depth research papers, 2 on a quantum algorithm and 2 on a quantum topic, so once the course is over, students can continue to apply those skills attained through this research and continue to hone their knowledge, even if it's not necessarily in Quantum Computing Fields.

For this portfolio, I will be highlighting my research paper on Quantum Error Correction, though if interested, my papers on Grovers Algorithm, Simons Algorithm and Quantum Entanglement will all be available at <https://nelsontorresjr330.github.io/> .

While the idea of Quantum Error Correction is mentioned in the course, no emphasis is placed on it and all that's explained is that it is one of the fields of Quantum Computing that developers and researches are actively trying to hone. My paper on it explains what Quantum Error Correcting is, why it is needed, and how it is currently being implemented.

All papers for the course follow IEEE conventions as shown below in Figure 5.

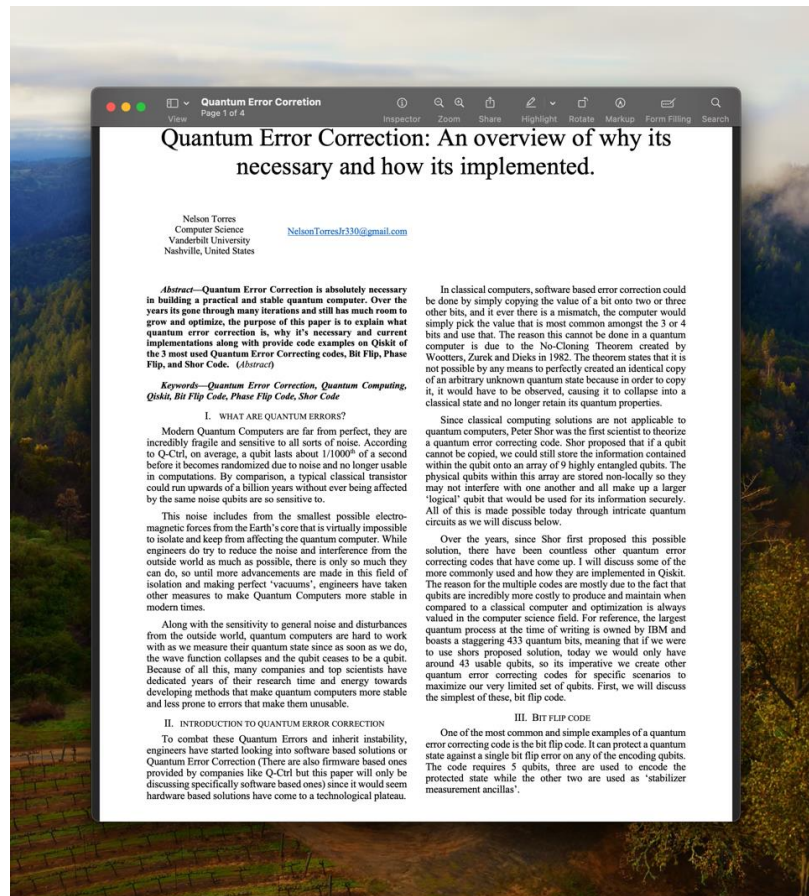


Figure 5: Quantum Error Correction

Due to the IEEE Convention, the papers always start with an abstract, a keywords section, utilize Roman numerals to split up the body, and end with a conclusion paragraph and references cited, as shown in Figure 6.

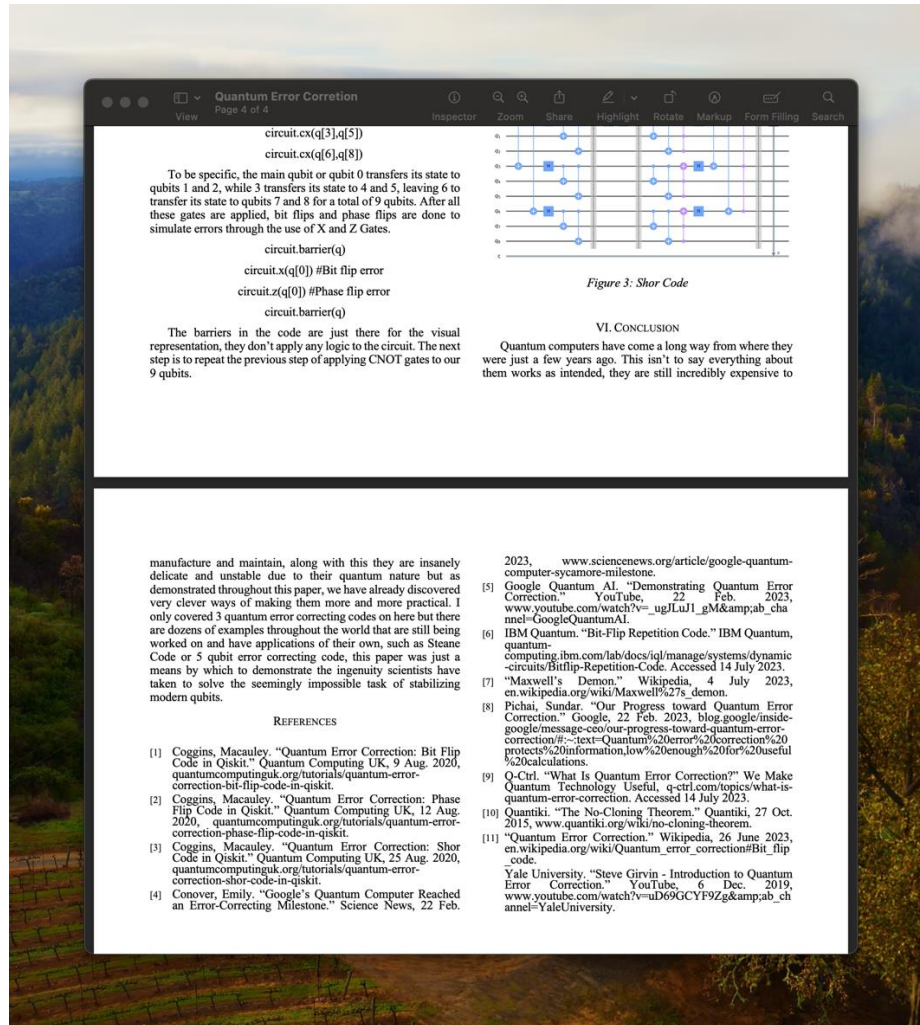


Figure 6: Conclusion and References

Another note to mention is the need for code snippets and visualizations where applicable as shown in the top half of the screenshot from Figure 6. Without getting too deep into the weeds, this specific paper required I research and fully understand various complex Quantum Computing topics outside the scope of the course such as the No-cloning theorem, Maxwells Demon, Quantum Decoherence and Classical Error Correction.