Zach Rankin

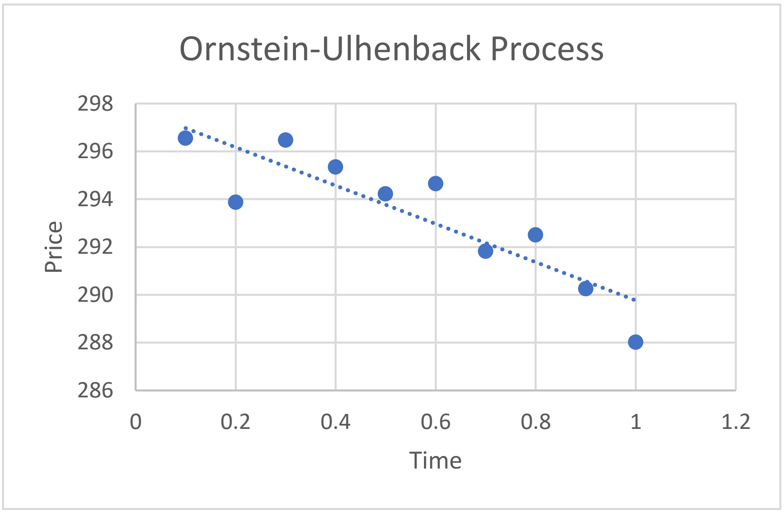
CS480 – Senior Project

3 December, 2025

Final Project Paper

For my final project I made a video game centered around making money through a couple if different ways. The story revolves around the player being a corrupt government official who is trying to pay back their debts through any means available. The two ways to earn back their debts are betting on the stock market and through taking bribes to vote for different bills. These two are combined with a simulated email client to receive communications from companies offering bribes, and with a place to pay the debts accumulated by the player. I have a minimal story in place, where there is one explanation email sent to the player and then they only receive different bribe offers. I wanted to have a sparse story going in to sort of make the game feel like the in-game character is isolated to almost encourage more unethical behavior on morally gray issues, like taking bribes.

My favorite algorithm used in the project was a mathematical one that was adapted to model and demonstrate values in a simulated stock market. When I went into making this section, one of the main things I wanted to have was something that did not look like a random number generator picking values out of a hat with no concept of the previous values. I wanted to get something that demonstrated values that a trend could be gleaned from, and if a player wanted to they could bet in accordance with that trend. After looking at quite a few different possibilities, I settled on using the Ornstein-Ulhenback process to model the market. One of the main hurdles with this process is that it uses more advanced than I have taken, meaning I needed to use a modified version to implement it. The original equation is: $dx_t = -\theta x_t dt + \sigma dW_t$; however, I do not know how to deal with, nor code, differential equations, so the final thing I ended up using was: $x_{t+1} = x_t + \theta(\mu - x_t)\Delta t + \sigma\sqrt{\Delta t} * Z$. This version was a changed to be adapted for use in financial simulations, and is something I can confidently approach. I broke it up into its three pieces, with each doing a different action on the end result. These pieces are incorporating the previous value, the pull toward the mean, and some randomness. The first piece adds the previous value back in to maintain consistency. The second uses mu as the long-term trend value, theta as a pull toward that long-term value, and the total change in time to progress the values. The final piece adds in some random noise to the value to give it some deviation as it progresses toward the average. The value of sigma is how volatile each resulting random step will be, and Z is a Gaussian random number to add in additional noise and to determine the sign of the step. After compiling all of these parts each of my fake stocks have a decent value of randomness that helps to accentuate each of them and



makes it not only more interactive for the player, but also to make the stock system feel more realistic.

My favorite data structure throughout coding this game was a dictionary. Through doing a couple test applications, and doing a few exercises to start learning the language, I came across the built in one that Godot has, and it was amazing to use. In almost every script I wrote, there would be a dictionary in there somewhere. One issue that I came across with how Godot works is for making classes that are not associated with an on-screen node. It was possible to make them; however, it was more of a challenge to organize and implement them than having objects built into each file. As a result, I ended up going with making each needed object in the file rather than having a separate class per item. I like how objects with properties are in JavaScript, and I used dictionaries to emulate this same sort of functionality in game. One thing that made me really love the dictionary is how disambiguous it made my code, especially to debug. Where I would have normally had made a mess of array indices and many variables, I was able to condense and efficiently use the same object to collect and organize the data I needed, alongside giving each a semi-unique label. On top of using them to store the data while the program is running, Godot's built in JSON library had pre-made methods to take in dictionaries and JSON code and swap between them. Since I planned on using JSON files to save the game after each in-game day, this functionality of a data structure I already liked and was already using made making the save and load code much easier to write. Overall, using the dictionary shrunk my code by quite a bit and kept all of my objects organized, making it both my favorite and most used data structure throughout the run of my program.

The languages I used to code my project were GDScript and C#. Going into this project, I knew I liked C# and I wanted to have the option to fall back on that if I needed or wanted to. That being said, I wanted to learn something new as well. My thinking was to prioritize the new language and only use C# for any features I would have major problems with writing in something else. Wanting C# support in the engine limited me to a handful of possibilities, and after consultation, I decided to use Godot. I took the game design course offered during last semester, and in that class many of my classmates decided to use Godot in lieu of the Unity engine that the class was being taught in. Seeing how well it seemed to work for these people accompanied with advice to use it from my committee head, Jason Haskell, made it an easy choice to use. Starting, I was intending to have a near eighty to twenty split of GDScript to C#; however, despite Godot having support for C#, it is not very good support for my use case. Trying to connect any onscreen action to a script was done seamlessly through GDScript, and was a tremendous headache through C#. It ended up being that the vast majority of my project was written using GDScript, having only a little bit of C# tacked on for one part. That is to say, I like using GDScript and it made writing my project a fun challenge to accomplish.

I did really like using GDScript and C#, but that being said, I would not use them for this kind of project again. There were many different problems that made it difficult to approach using Godot for this kind of game, and in future, I believe that it would be better to make this using a different set of languages. Through taking the web programming class it become more and more apparent that using something like Angular or Vue would have made this project easier, and would have provided more options to better customize and develop this game. I can see the benefits of using game engines for making games; however, with my game being largely menu based, with all inputs being done via mouse clicks, Godot was not the best fit I could have used. I originally envisioned this project as a modular type of application that had each feature wrapped up in its own little area, and with each feature being able to be plugged in or removed whenever was needed. This functionality is much less apparent through my use of Godot, especially compared to something that was made to have black box functionality. On top of the design quirks of Godot, having the different hurdles of incorporating outside C# classes made it much less manageable to deal with. Through this project I believe that the original design that I had in my head would have been better implemented through a different context.

During this project I did not use any external libraries, and though it did make it slightly easier to understand in the sense that I knew how my own code worked, by not using them I was limited by time and personal knowledge. One area in particular I could have benefited from using an external library was through making and using graphs. In the stock section of my game, each individual stock displays a graph of its most recent data points. I achieved this through overriding a standard node's draw property and then computing where to place each and every point and line. In my implementation, I manage to get a back-grid line pattern, axis, all of the points, and a line connecting to each point. I am able to display the data and make it useable and readable to the user, through having a dedicated comprehensive library would have made this have more features. I believe a dedicated graphing library could have added more functionality and allowed me to add extra features to the graph itself.

There are many things I would do differently if I were to do this project again. These include making it on a different platform, organizing the elements differently, and focusing on different aspects of the design. As said above, if I was going to remake this game I would do so in a web context, and on top of that, I would change what to implement in it. Currently there are three save slots available, and each of those are based off of a JSON file per slot. This works for my small set of data to save now, but if the project were to continue being developed I would need to change how that data is used. Currently I load a file and pass the chunk of data around to each of the big scripts with each one taking any piece that it needs. I would switch this to being divided up more if I were to make this again, probably having a unique JSON file per page and having it manage its own data per save and load.

A second thing that could be done differently was to change how I approached each design. This would be mainly about reworking the current systems that have to be there, and better focusing on adding more functionality rather than tweaking one difficult system. The main one I would rework is my current version of an email client. I wanted to have some way to communicate story elements to the player, and have the ability to send bribes for the player to take or reject. After making the email app I designed, it does work to get information to the player though not quite as well as I would have liked. I was limited by both physical design and by the design limitations of my code. Thinking back on the design, I wanted to have a one-way communication and making a pretend email client was not the best avenue to go about doing that. If I was to remake this, I would change to having a design with less available options, in the sense that it could be more of a messaging platform. This could have a different panel per message and collect messages in each one for the player.

A different system that I would rework, and subsequently spend more time on would be to make the story more integrated into the design of everything. I approached this project by getting everything working and then adding in a story after the fact. This way of doing it did work, but it did cause many issues down the line. To rework some systems, like the email client, to have the story built into it could have made the design shine more. My system works decently well to relay information to the player, but I would like to make it more interactive and incorporating the story as a design element would have helped that come together better. If I redid this, baking the story into my design would be something I would focus more on.

On top of those changes, I would change how I divided my time. Going into the project I was envisioning having many different avenues for the player to go down to make money, and having the decisions of the player effect how the game played out. I wanted to have their decisions have real consequences, and be able to make reactive changes; however, this takes a decent amount of time to accomplish, and this was time I did not have. One area I feel I should have cut down on time was in the stock page. It took a while to make the graphs look good, and a long time to find, understand, and implement an algorithm to calculate each stock's data. In the future, I would spend less time on this sort of thing and focus on adding more features to the project as a whole. Whether that would entail cutting parts of the stock page or through making less options available, I believe that it would make the project feel more cohesive if there was a better spread of time spent on each individual section of it.

My feelings about this project are overall quite good. I like my project, and I thought the idea was fun to play around with. I feel like I did a good job in making it. At the start of the project, the general goal to aim for was about one thousand lines of code written, and with than in mind I feel I did a good job. I like how the code works, I am happy with the features I have built into it, and I think that it was a challenge to accomplish. I have not had very much experience making games before and that inexperience show me how much time gets invested into making a project like this. Despite taking the game design course, I had many hurdles to overcome to progress through the project. One of these hurdles having to both learn how to code using Godot and GDScript while forging ahead on the project. It was a hard to overcome and I fell that it changed how I will approach problems like this in the future. After completing this project, I can certainly say I am proud of what I accomplished and happy with how my project ultimately turned out.

Zachary Rankin

Marquette, MI 49855 | (906) 282-5027 | zrankin@nmu.edu | linkedin.com/in/zachary-rankin-a84092243

Motivated Computer Science student skilled in programming, web development, and game design with experience in Python, Java, and Unity.

EDUCATION

Northern Michigan University — B.S. in Computer Science
Marquette, MI | Aug 2022 – Present
GPA: 3.97 / 4.00 | Key Courses: Data Structures, Algorithms, Software Engineering, Database Systems

RELEVANT EXPERIENCE

Computer Science Tutor — Northern Michigan University
Aug 2024 – Present

- Assist students in Python, Java, and C++ across core courses.
- Support debugging, algorithm design, and problem decomposition.
- Adapt explanations to diverse learning styles, fostering collaboration and confidence.

OTHER ROLES

- Physics Tutor, Northern Michigan University, Aug 2024 – Present
- Groundskeeper, City of Iron Mountain, May 2023 – Aug 2025

TECHNICAL SKILLS

Languages: Python, Java, C++, C#, C, Swift, JavaScript, TypeScript
Frameworks & Tools: Vue.js, Angular, Node.js, Godot, Unity, Git, Agile Development
Core Competencies: OOP, Data Structures, Algorithms, Full-Stack Development
Operating Systems: Windows, Linux, macOS

LANGUAGES

English — Native | Spanish — Intermediate (Classroom Study)

INTERESTS

AI, Game & App Development, Software Design, Full-Stack Web Development, UI/UX, System Architecture