

Portfolio

My name is Joseph Hamilton, and I am a freshman at UL Lafayette living here on Campus.

Academics and Internships

I attended a Computer Science Camp at George Mason University and was rewarded with a 1 College Credit that transferred as an Electrical Engineering Credit. The Camp was purposed around game development, and they taught us how to use Unreal Engine and how the game development career looks like.

I interned at the IT company, *Techneaux Technology Services*. I was there for 1 summer and shadowed the Software Development team. I was shown how to use Razor and ASP.NET with the C# programming language.

Experience

I have been coding for 5 years, since my freshman year in high school. I started out with learning Game Development and C#. After a year and a half with C# I started to really understand this language and branch out to new languages. I got into HTML, CSS, and JS with Web development for a bit, and made my dad a website for his company using HTML code. I then started to get into Rust and made a Programming Language, Scone. I never completed this language. After this I began to love using C and I made another language galo, but got lost in memory errors and never finished it. I really love working with lower-level languages and learning how stuff works. Recently I've been working on a custom WebAsm Assembler and Parser. I find it the most fun working on lower-level computer topics. This WebAsm project has taught me about Opcodes and using the stack and passing arguments via pushing to the stack. I had made a similar project in Rust a few years ago, but this is a much better rendition of it by actually writing my code in WebAsm opcodes and JS.

Links

Here are a few links to some of my stuff:

- Website: jbros-development.web.app
- Coding Projects: github.com/JBrosDevelopment
- Games I've made: jbrosdev.itch.io/

Projects

Here is a list of some of my projects I've worked on, with my favorite at the top:

- **WebAsmCompiler:** github.com/JBrosDevelopment/WebAsmCompiler

- This project follows the evolution of computers from opcodes and commands to compilers. I'm keeping up with my progress in the README of this repository, right now I'm working on the assembler. I can only make the assembler from the Web Assembly opcodes, and then I can only make the compiler from the assembly language earlier created.
- Surprisingly, WebAsm is straightforward and much easier to work with than native x86 assembly.
- **MenCwaft:** github.com/JBrosDevelopment/MenCwaft
 - This project is a Minecraft clone that has both a client and server structure using WebRTC and Firebase as a signaling server. I have to admit a lot of this project was vibe coded because I didn't know how to use the THREE.js library or WebRTC. That being said, I could tell you how every function works.
 - You can try it out here and create and join a server: jbrodevelopment.github.io/MenCwaft
 - There is no server in the cloud; the tab you open to create a server is literally acting as the server between any client tabs you open.
- **galo:** github.com/JBrosDevelopment/galo
 - Galo is an attempt at a compiler and interpreter made completely in the C programming language. I left this project unfinished when I got bored of it after working on it for so long, but a simple program will run in the interpreter. It has a full CLI and a Lexer and Parser.
 - The galo language syntax looks like this:
 - ```
fun add(left int, right int) int
 return (left + right)
end
let x int = add(1, 2)
```
- **calc\_lang:** [github.com/JBrosDevelopment/calc\\_lang](https://github.com/JBrosDevelopment/calc_lang)
  - This is another programming language that is finished and works great. It is written in Rust and mimics the TI84 calculator language. It looks like this:
    - ```
max(X, Y) => X > Y: X; Y ... Function declaration with
                                ... `=>` and if else patterns
                                ... with `:` and `;`
3.14 -> pi                    ... lowercase is constant
15^2 -> X                     ... uppercase is variable
Xmax(150, X) -> Y              ... this evaluates to (X *
                                ... max(150, X))
```
- **VirtualComputer:** github.com/JBrosDevelopment/VirtualComputer
 - This Project was my first stab at going low-level in a computer system. Using Rust I made a Virtual Computer that takes in bytes and performs the commands related to the opcodes. It also has a small assembler and compiler that doesn't work very well. Overall, the project was fun but not usable in any way.

- **scone:** github.com/JBrosDevelopment/scone
 - Scone is another attempt at a programming language written in Rust. It is also unfinished. It is a lot more polished than galo but I stopped working on it before I could get past the validation of the syntax in the interpreter. In theory if I finished this language, it would be fun to work in and is designed to be very object oriented and very powerful out of the box. I never finished it though.
- **CrossPlatformWiiRaylib:** github.com/JBrosDevelopment/CrossPlatformWiiRaylib
 - This was a fun project made in C that stabs at using MakeFiles and making a graphics system that supports two different platforms. Using the MakeFile you can select to build the project as an .exe with RayLib or as a Nintendo Wii executable game that can be run in the Dolphin Wii Emulator. It has limited ability, but it was more of a proof of concept.
- **Chess:** github.com/JBrosDevelopment/Chess
 - Chess Engine made in C.
- **Norma:** github.com/JBrosDevelopment/Norma
 - Programming language made in C#
- **TerminalEngine:** github.com/JBrosDevelopment/TerminalEngine
 - A C# Package that makes it easy to use the console as a canvas to display pixels

I have a few more projects, but these are the main ones to note.

Language Experience

Here is a list of the languages I've been able to use and become comfortable with:

- **C**
 - I use this language a lot when I want to get down to the barebones of the computer.
 - I know how **C++** works, but I have not used it enough.
- **Rust**
 - I've used this language for years and I am very comfortable in it and love the way it enforces the code to be perfect
 - If I had to pick any language to work in for the rest of my life, I would choose Rust.
- **C#**
 - This was the first language I ever used and have a lot of experience with it mainly with Windows Forms and Unity Game Development.
 - I've heard **Java** is like **C#**, but I've never used **Java** before
- **JS**

- This is not my go-to language, but I do use it on occasion with Web Development, along with **HTML** and **CSS**.
- **Python**
 - I've never explored this language to its very depths, but I know all the basics with classes and functions. I use this language when I need to test a proof-of-concept because it's easier to "think" in this language and write code faster.

Language Experience

Throughout my years of learning programming, I have learned a lot of core programming concepts and “ways-of-thinking.” I will try to list a bunch and explain it in the best way I can.

- **Object Oriented vs Functional**
 - In **Object Oriented Programming**, you think of every variable, function, and value as an “object.” Every Object has a type associated with this object.
 - **Functional programming** is a different way of programming where every line evaluates a value. I don't have too much experience with this, but I know languages like Haskell operate in this way. One thing a functional language has is the lambda functions.
 - **Multi-Paradigm** languages are programming languages that allow both object-oriented and functional programming in the same language. Most of the most used languages operate in this way.
- **Classes** are a way of defining a way a data structure should hold its data. On top of that, classes have methods inside of it that can affect the class when called.
- **Constructors and destructors** are used in defining classes. Constructors are like other methods in a class, but they get called whenever a class is instantiated. Destructors define how the class is disposed when the garbage collector cleans up the object.
- **Inheritance** is a way of programming where you can define a class based on another class. For example, you can have a `dog` and `cat` class, but each of them can inherit from the class, `animal`. This is useful if `dog` and `cat` both have a `walk` method and you have a function that can take any type of `animal` and call the `walk` method.
- **Abstraction** is used when defining a class that acts like another class, so that when defining the new class, you don't need to define unnecessary functions and rewrite code. It is like inheritance, but the main difference is that abstraction aims at removing the need to rewrite functions.
- **Interfaces** are a sort of blueprint for how a class should be defined. It explains what a class can do without defining it specifically.
- **Polymorphism** is the basis that the same function calls can act differently depending on the object. Say there's two objects, one with the type of `dog` and one with the type of `cat`. Say they also both inherit the class, `animal`, and have the

function, `speak`. They each have different implementations of this function. Say `cat.speak()` could print "meow" while `dog.speak()` could print "ruff". Polymorphism is the concept of this type of programming.

- **Compiler vs Interpreter**

- A **compiler** will take your code and output a binary file, often as an executable, to run directly on your computer whenever. C is a great example of this.
- An **interpreter** is a separate program that will take your code and execute it line by line. Python and JavaScript are great examples of this.
- There is a sort of combination with a different approach using JIT compilers. These hybrids usually have compilers that compile to lower-level code that the runtime will interpret. Java and C# are examples of this.