CS 1632 - DELIVERABLE 7: Rock-Paper-Scissors in Rust

Michael Bowen

Summary

As expected, the primary difficulties encountered for this deliverable arose from being new to the Rust programming language. While I am relatively comfortable programming in C, the differences between the languages required a little bit of research and some forethought when developing the sections of the code.

Meeting the functional requirements of the deliverable was easy enough, as the flow of Rock-Paper-Scissors has been familiar since childhood and the logic of the game lends itself easily to developing a program that can emulate it. The real challenge for the deliverable was rooted in the implementation requirements. Those details, due to the idiosyncrasies of Rust, ended up being just a tad complicated, but nothing that couldn’t be handled.

One challenge that I faced was with the enums. At first glance, an enum consisting of Rock, Paper, and Scissors seems to be enough. However, due to how I handled the user input, the Choice enum ended up needing two additional values: Invalid and Quit. There is probably a better way with which to handle this, but I felt that folding those two values into the Choice enum made some sense and allowed for the use of a match statement for handling all of the user’s input. It did complicate the other match statements that centered around the Choice enum, however. When I went to generate the computer’s choice for rock, paper, or scissors, I ended up forgetting to handle the other two options at first. Understandable, since the chance of the random number generator giving an incorrect value is fairly inconsequential and thus, the computer will never choose Invalid or Quit. Luckily, the Rust compiler is very good at pointing out the problems and this issue was easily solved by implementing an extra match clause that handled the other possibilities.

The other bigger problem that I ran into was when trying to ensure that one of the functions correctly used borrowing. I had first attempted to do so with the user’s input, but was getting a type mismatch error that, at the time, had me a bit stumped. So I moved on and got the rest of the program implemented correctly. The functions that are used to update the data contained within the Statistics structure use borrowing to manipulate that data, but I wanted to use borrowing in at least one other instance incase those weren’t truly borrowing. So, I reworked the get\_cpu\_choice function to borrow the variable and update it within the function. At first, I got the same type mismatch error I had been getting previously, and got a tad frustrated. It turns out, that in my haste to get the new Rust syntax correct, I was forgetting my C and simply wasn’t dereferencing the variable correctly.

The URL for the code is below. I decided against attempting any of the extra credit simply because I don’t have air conditioning and the laptop is hot. But it would be interesting to revisit at some point as Rust seems compelling. Presented is a functioning Rock-Paper-Scissors game free of compiler warnings.

Code for the deliverable: github.com/mjb236/CS1632/tree/master/Deliverable%207