II. Background

2.1 Overview

Chapter 2 will give an overview of the necessary information to follow this thesis. This overview will provide the fundamentals of the Rust programming language. With that, basic networking, specifically with Rust, will be explained. Next, the Entity Component System (ECS) architecture, and an implementation, Specs, will be described. Furthermore, the physics for flight dynamics modeling will be laid out. Finally, the flight simulator, FlightGear, will be explained.

2.2 Rust

Performance, concurrency, memory efficiency. Rust is a fast programming language – matches r even exceeds speeds of its peers. “fearless concurrency” parallel programming is free of errors that other languages have which makes concurrency be seen as difficult. “zero cost abstraction” features gained from rust have no runtime cost, safety does not mean a slower program speed. Downsides include longer compile time, compiler is strict, large language [1]. Compiler is like a safety net. The type system is smart enough to find all the urn time bugs. More time to compile, but once it does, its most likely right. Use rust for correctness. Good type system. Can be sure code is correct at compile time. Rust has no garbage collector, the compiler is the garbage collector. Ownership replaces the need for a garbage collector and assures memory safety. Borrowing references avoids data races with two pointers accessing the same memeory at the same time. This is good for performance, and less memory overhead. Community is excellent, ecosystem. Mozilla has been the driving factor behind rust. Used in firefox. Ecosystem is quickly growing, providing libraries, called crates in rust. Rust model pushes you to design your programs differently than OOP. Programs Less about execution flow and what the cpu is doing and more about how data is laid out in memory. And how data is laid out in memory . and how ownership of memory is given to different parts of the program at run time. Spend more time getting the compiler right and less time fixing bugs.

Ownership, borrowing (references), lifetimes = memory safety. There is exactly one binding to any given resource. When ownership of memory is transferred to another binding, the original binding cannot be used. Only one reference can own a piece of data at a time. References allow borrowing a piece of data for a moment [2].

Clear up these thoughts on ownership   
  
Traits…Error Handling…

2.3 Networking and Rust

Networking can simply be defined as two programs communicating with each other, sending and receiving data. To do this, two sockets are required at each end of communication. In this case, the type of socket being used is the Datagram socket. The protocol involving Datagram sockets is called the User Datagram Protocol (UDP). This protocol is considered “connectionless” because an open connection does not need to be maintained. Packets containing bytes of data may simply be built and sent to a destination without a connection needing to be established prior [3].

In Rust, a UdpSocket can be created and binded to an ip address and port on the local machine. After this, data can be sent and received from any other socket address. The socket created can then be connected to the other socket with its ip address and port. From here, data can be sent using the send function, and data can be received using the recv function. (code example from rust-lang)

A machines computer architecture represents bytes in a specific order. There are two byte orderings: most significant byte comes first in the ordering, and least significant byte first comes in the ordering. Most significant byte first is called Network Byte order, and least significant byte first is called Host Byte Order. When sending and receiving data, one must be aware of the order on either machine which is necessary to interpret the bytes correctly. So the order may need to be converted first before packets of data are sent from one socket to another [3].

2.4 Entity Component System

The ECS architecture is a viable replacement to Object Oriented Programming (OOP). Where OOP focuses on hierarchy, ECS focuses on composition. ECS compartmentalizes Systems from the data, and Systems from other Systems. This architecture is powerful because it decouples code, making maintenance and editing code easier. The goal is to avoid hierarchy which causes hierarchy chains and code too highly dependent on other code [4].

OOP has Classes and Objects, and ECS has Entities and Components. This does not mean and Entity is the same as a Class, and this does not mean a Component is the same as an Object.   
  
Entities are the building block of the system, and each represents a different in-game object, such as “5 tanks.” Entities do not have any data or functions. They are useless on their own. They are containers which components can be added to.  
  
Components are the data representing Entities. All in-game objects have multiple aspects, or attributes about that object. For example, bicycles are made of metal, and are a means of travel. This data can be added to an Entity, labeling the Entity as possessing a particular aspect.   
  
Systems deal with the global stuff outside of the realm of Entities and Components.

<http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/>

<http://entity-systems.wikidot.com/>

2.5 Flight Model Physics

2.6 FlightGear

References

1 Rust in action book  
2 <https://doc.rust-lang.org/1.9.0/book/>  
3 Beej’s Guide to Network Programming **-** Using Internet Sockets  
4. ECS game engine design