Forest C Sharp
VUNet ID: SharpFC
Peer-to-Peer Networked Gaming With Python and UDP

Introduction

For my final project, I have implemented a simple Peer-to-Peer (P2P) networked game using Python, UDP sockets, and the Panda3D real-time rendering engine.

Motivation

Starting this summer, I'll be working at a game studio that employs Python for much of their networking backend. Although I'll mostly be doing gameplay programming in C++, I was excited by the opportunity to learn how to use sockets in Python as well as brush up on my Python programming skills in general.

Design

There are four main files that hold the bulk of my project's code.

First, main.py creates a subclass of Panda3D's ShowBase class called App. This App class is responsible for the setup of the local player, opponent, and P2P connection (all of which will be discussed on below). Additionally, below this class in main.py are two function calls that actually instantiate and run the app.

Second, P2PConnection.py is responsible for the network connection between the two combatants. Upon creation of the P2PConnection, a UDP socket is set up for both the sending and receiving of UDP Packets. The StartConnection function initiates a UDP Listener thread to process messages received over the GroupCast IP. Additionally, there are two register functions for providing references to the local and opponent player objects.

Third, MyPlayer.py represents the local, controlled player of the client. As the user supplies input to MyPlayer, the necessary data is sent via the UDP protocol over the network to the opponent.

Finally, NetworkedPlayer.py represents the opponent. The NetworkedPlayer's movement and behavior is entirely governed by the received UDP Packets. The NetworkedPlayer is a simulation of the opponent client's MyPlayer.

Types UDP Messages

There are three types of UDP packets sent over the network, identified by their OPCode (the first argument of the UDP packet). Depending on the OPCode, the UDP packet then has several more arguments, and finally ending with a playerID, unique to each player. These three packets represent Movement, Attacking, and Destruction, respectively. They are outlined in the table below.

The movement operation, which is by far the most complex, has 6 primary arguments. These represent the X, Y, and Z coordinates as well as the Heading, Pitch and Roll of the entity.

The attack operation simply sends a damage value.

Finally, the destroy operation has no arguments other than playerID, and is simply there to signify that one player has lost and the other has won.

OP_POS	X	Y	Z	H	P	R	PlayerID
OP_ATK	DMG	PlayerID					
OP_DES	PlayerID						

Future Considerations and Improvements

It would be nice if the game could support greater than two combatants, and initial support has been added for this, as each UDP packet is also identified with a PlayerID that could be used to identify greater than just 2 players.

I would like to implement a better PlayerID system as well, so that it doesn't just generate a playerID as a random number. Although the range for my random integral playerID is currently quite high, two players could technically still have the same PlayerID, however unlikely that may be.