GPR-430 Networking for Online Games
Networking Final Project - Planning Document
Scott Dagen, Ben Cooper
2021-03-09
Professor Ashish Amresh

Goals

Networked Audio (ideally synced music across a network)
-- Not testable in a remote environment, scrapped

Networked Game State, primarily client authority

Mechanics
WASD = movement, IJKL = shield positioning
Space = shoot

Bullets must bounce off of shield to be able to destroy enemies
Enemies do damage to pillar on contact
Players absorb bullets without taking damage

Stretch Goals
Item can be picked up, can heal player or give fire rate boost
Bullets have knockback in the direction they're moving
Procedural levels (goal is to reach the end of a level with a given seed, determine length of level from seed)

Implementation

Networking:
Client broadcasts its own player state, bullets created by that player, and half of the AI
Server broadcasts timer events and pillar damage, as well as lobby info

Messaging:

We rely on Marshaling to convert data to and from bytes, as explained in https://stackoverflow.com/questions/3278827/how-to-convert-a-structure-to-a-byte-array-in-c, and we add a byte at the front to store the message type. Message handling is accomplished via switch case.

Dead Reckoning:
When the AI or remote player receives state information, it checks if the state is too far from its current position. If it is, we snap to the new position, otherwise we lerp towards it over time. If the passed-in velocity is facing too different a direction (30 degrees+), we snap to the new

velocity, otherwise we lerp to the new velocity over time. The bullet behaves similarly but velocity can only be in one of 8 states. As such we only need to lerp the position.

When running the dead reckoning algorithm, we continue to simulate where we think the next packet will send the AI by updating the “expected position” with the last received velocity every frame. This means that we rarely have to actually snap as we’re usually close to the next point that’s sent.

Async:

The lobbies behave similar to the checkers implementation from Project 2. Multiple validation steps take place and clients mostly wait until everyone has connected before the game can start.

Assignment Description

Instructions:

For the final project, you will design and implement a portfolio-worthy tech demo, game or tool that demonstrates and integrates all of the course subject matter in some meaningful way. Consider the following example base designs for your project, which are in no way the limit of what you can accomplish:

- Implement the complete networking architecture for your Capstone or Game Studio/Production game, either by integrating a custom plugin wrapping RakNet, or using minimal existing engine features;
- Implement a tool that helps facilitate real-time networking for games, or uses real-time networking to achieve some meaningful purpose, such as something that helps programmers debug remotely;
- Implement a demonstration of an advanced topic of interest, something that is a new, creative and interesting application of any set of the course topics integrated in a meaningful way;
- Implement a demonstration of a novel system or framework that improves real-time networking development and workflow, such as replacing RakNet with your own networking wrapper.

All components of the project must be functional and impressive; this is something you would want to show off at an interview... hence, this project is tied with a technical interview! Find a way to demonstrate the requirements as part of the technical interview (which involves a presentation). Implementing the bare minimum will earn you up to 80% on the project; go above and beyond for the remaining 20%.

Requirements:

Design requirements: These count towards the organizational part of the project:

- You must have a thoroughly organized repository with branching, and evidence of the codebase evolving over time and absolutely not slapped together in the last week.
- Your repository must include a read-me and any design documents, such as UML diagrams and/or technical specifications, that are started early in the project and clearly referred to throughout development. These documents are strictly focused on the subject matter of this course; e.g. if you are working on the networking of your Capstone, we don't care about game design documents or graphics, we care about the networking.

Project requirements: These count towards the implementation part of the project:

- The project must include and integrate some elements of each of the main course units in some meaningful way:
    - Asynchronous architecture;
    - Synchronous architecture;
    - Remote agents and synchronization;
    - Remote simulation and correction.
- You must include your own implementations of the main topics covered in each of the above units. Using the engine or networking framework of your choice (e.g. RakNet, Unity or Unreal) should mean stripping their implementations down to the base basics and working from there. The bare basics include connection, disconnection, sending messages and receiving messages; any framework is certain to give you access to these fundamentals. Working from there, some examples of the core topics in the course are as follows:
    - Server/client architecture with each peer given their own distinct responsibilities for optimization:
        - The server's architecture is responsible for simulation, corrections and management only, and should not be able to render in real-time or receive direct input.
        - The client's architecture adds the primary responsibilities of handling user input and real-time display.
        - Ensure you also have a common interface or library that manages the common elements of the server and client; e.g. knowledge of the game state, game data, bi-directional messages.
    - Include user-controlled agents (PCs) and autonomous agents (NPCs where appropriate) that can be simulated and efficiently tracked remotely.
    - Include methods such as dead-reckoning, interpolation-based and multi-step simulation correction and synchronization techniques.
    - Include data packing and unpacking methods when dealing with large chunks of networked data.
    - Use object-oriented and ECS design patterns where appropriate, using memory management principles to maximize efficiency.

    - **These requirements may be updated and clarified over time to best reflect the progress of the course and lessons**.

Presentation requirements: These count towards the demonstration part of the project:

- In your video, you must thoroughly demonstrate the outcome of your project and explain how all of the above implementation requirements are met. Walk through and summarize the main architectural contributions in code.
- You should not explain every little thing that you implemented; this is a 10-minute summary so please figure out how to get to the point and share only what needs to be shared, and thoroughly demonstrates your prowess as an engineer in this area.

Extra Credit:

For teams of 1-2 people, completion of the following will be optional and yield bonus points; for teams of 3, these are added to the project requirements and will not be used to give bonus points:

- Pre-game or post-game state such as an advanced matching or tournament system, or a leaderboard;
- Admin client to interface with the server;
- More TBA but you can propose your own bonus features!