Network Programming – Game Design Document

Gameplay Experience

The gameplay of the Tanks inspired game is about strategically aiming ricocheting bullets to hit the other player(s). The players will respawn after a couple of seconds, this will continue until a player reached the win-condition: the first to hit 5 kills.

The challenge of the game is successfully hitting the other player while evading the bullets they are shooting at you.

Art Direction

The game will have a top-down perspective. The art style will be pixel-art with a retro feel to it: vibrant and simple.

Multiplayer

The multiplayer aspect of the game appears in the players all controlling a tank, and all having the ability to move and shoot at other players, affecting the other's game state.

The game will automatically find a server running on LAN and query its own ip address.

Entities

- Tank: Controlled by players
    - Vector2 Position
    - Float Rotation
    - Bool isAlive
    - Float Velocity
    - Float AimRotation
    - Timer RespawnTimer
    - Timer ShootingCooldown
- Bullet: The direction to shoot at is controlled by players
    - Vector2 Position
    - Float Duration
    - Float Velocity
    - Int RicochetCount
- Wall: Obstacles that cannot be crossed or broken down
    - Vector2 Position
    - Float Width
    - Float Height

Requirements

Pass Requirements

- *Networking and Architecture*
    - All network communication between the client and server must be handled using UDP
    - The game must use a *client/server architecture* with an **authoritative server**
        - All critical game logic and state changes must be validated by the server.
    - You must implement and use a simple *virtual connection protocol* on top of UDP to manage the connection state for all clients.
        - This protocol ***must*** handle
            - A handshake process using ConnectPacket and DisconnectPacket to establish and gracefully terminate connections
            - Game state data should be appended after PayloadPacket
    - Implement a system to serialize multiple, variable-sized game messages
        - This for both client-to-server and server-to-client communication
        - E.g., PlayerPositionMessage, SpawnBulletMessage, ObjectPickup Message, after the PayloadPacket header
    - Limited to **1024 bytes** of data for a packet
        - This includes messages that are potentially sent each send-tick (see byte_stream)
    - The server must send game state updates to clients at a consistent rate between **5-20 packets per second,** i.e. use a fixed send-rate
    - The server and client should detect when a connection has timed out and handle it gracefully
        - Time-out when no packets were received for a set duration (e.g. 5-10 seconds)
    - The client must calculate and visualize its round-trip time (RTT) to the server

        - Value must always be displayed on the screen in milliseconds (e.g. "RTT: 5.72 ms").

- *Game Simulation and Feel*
    - The game logic on both the server and client must run at a fixed **60 Hz** simulation rate with a fixed time-step i.e. tick-rate
    - Remote entities must move smoothly on the local client's screen
        - You must implement ***entity interpolation*** to render these entities between two or more known server state updates
    - The local player's movement must feel responsive
        - Implement ***client-side prediction*** to move the local player avatar immediately based on input
        - When the server sends a reply the client should make a correction (***reconciliation***) if there is too big of a discrepancy between what the client predicted and the authoritative server position
- *Gameplay and User Experience*
    - Players must be able to perform at least one meaningful action that affects the game state and is communicated over the network
        - E.g. shooting a projectile, placing a bomb, or picking up an item
    - Network issues should be handled gracefully
        - If a client is disconnected or times out, a clear message (e.g., "Disconnected from server," "Connection timed out") should be displayed on the screen instead of the application crashing

PASS WITH DISTINCTION REQUIREMENTS

- *Server Discovery*
    - Find a server running on LAN automatically
- *Reliable Messages for Events*
    - Implement reliable messages for game events using acknowledge (potentially with some acknowledge history)