

Design Report for Spaceships Multiplayer Game over LAN (UDP)

# Written by: Thang Weng Khong (wengkhong.thang, 2301372)

# Irwen Yap (y.ziyangirwen, 2301345)

# Tan Yong Chin (t.yongchin, 2301359)

# Wilmer Lee Jun Rong (wilmerjunrong.lee, 2200691)

## **(a) Game Design Introduction**

### **Overview**

### The Spaceships project is a multiplayer game inspired by the classic Asteroids arcade game. The objective is to destroy as many asteroids as possible while maintaining accurate state synchronization across all clients over a LAN using UDP only. The game also supports single-player mode and handles communication robustness, reconnections, and score persistence.

**Multiplayer Objectives**

* Synchronize asteroid spawning, movement, and destruction across all clients.
* Synchronize player ship movements and bullet events in real-time.
* Implement reliable event propagation using a lock-step architecture with ACK tracking.
* Display real-time scoring and persistent leaderboards.

**Synchronization Model**

### Asteroids: Spawned on the host at fixed intervals and broadcast to all clients. Position, scale, and velocity are included in the packets.

### Bullets: Fired by players, converted into **FireBulletEvent**, and transmitted using the lock-step method.

### Collision Events: Detected only on the host. Once confirmed, a **CollisionEvent** is sent and committed across all clients.

### Score Updates: The host maintains and updates scores consistently across all instances.

**Network Design**

### The game uses UDP exclusively, implementing custom reliability over it. Events like firing, spawning, and collisions are sent using a lock-step mechanism where:

### The host assigns an **EventID**, broadcasts the event, and waits for acknowledgments (ACKs) from all clients.

### Only after all ACKs are received does the host send a **COMMIT\_EVENT** to authorize the action.

### Clients hold pending events and do not process them until the commit message arrives.

This ensures synchronization for deterministic objects and gameplay fairness.

## **A diagram of a server AI-generated content may be incorrect.**

## **(b) Implementation Block Diagram and Introduction**

**Command Identifiers (CMDID)**

The multiplayer protocol is built around a set of command identifiers (CMDID) that define the purpose of each network packet. These identifiers ensure clear communication between host and clients:

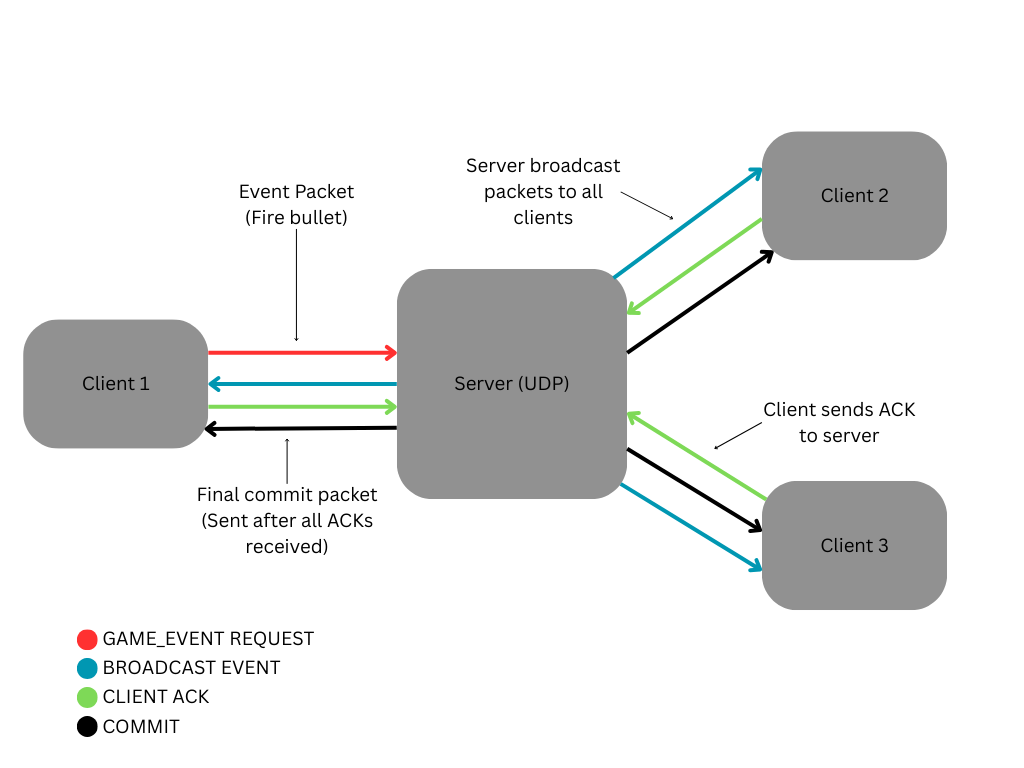
* REQ\_CONNECTION (0x1) / RSP\_CONNECTION (0x2)
  + Handles initial connection handshakes between clients and the host
* TICK\_SYNC (0x3)
  + Periodically synchronizes simulation ticks to maintain timing alignment.
* GAME\_DATA (0x4)
  + Transmits player state updates like position and rotation.
* GAME\_EVENT (0x5)
  + Sent from clients to the host to request gameplay actions (e.g., fire bullet).
* BROADCAST\_EVENT (0x6)
  + The host broadcasts events to all clients after assigning an Event ID.
* ACK\_EVENT (0x7)
  + Clients use this to acknowledge receipt of broadcasted events.
* COMMIT\_EVENT (0x8)
  + Host confirms all clients have ACKed an event and commands them to execute it.
* HEARTBEAT (0x9)
  + Keeps connections alive and detects disconnections.
* PLAYER\_LEFT (0xA)
  + Host informs others when a player disconnects.
* INITIAL\_STATE\_OBJECT (0xB)
  + Sends current game state to a newly joined or reconnected client.
* REQ\_RECONNECT (0xC) / RSP\_RECONNECT (0xD)
  + Manages reconnection flow and restoration of player state.
* FULL\_STATE\_SNAPSHOT (0xE)
  + Delivers a complete game snapshot to sync clients.

**Engine Structure**

* AsteroidScene:
  + Handles game object updates, asteroid spawning, collision checks.
  + Pushes and processes game events like **SpawnAsteroid**, **FireBullet**, and **Collision**.
* NetworkEngine:
  + Initializes UDP sockets using Winsock2.
  + Manages client connections, ACK timeouts, heartbeats, and broadcasts.
  + Coordinates lock-step synchronization via **BROADCAST\_EVENT** and **COMMIT\_EVENT**.

**Lock-Step Synchronization Flow**

1. **Client Action**: Client sends a **GAME\_EVENT** (e.g., fire bullet) to host.
2. **Host Reception**: Host assigns an **EventID**, stores the event, and broadcasts it.
3. **ACK Tracking**: Host tracks acknowledgments from all clients.
4. **Commit**: Once all ACKs received, host sends a **COMMIT\_EVENT** with a network ID.
5. **Execution**: Clients then process and execute the committed event.

****

**Data Reliability Techniques**

* ACK tracking with resend logic (**CheckAckTimeouts()**).
* Disconnection detection via heartbeat timeout (**CheckTimeoutsAndHeartbeats()**).
* Minimal delay due to LAN speeds (2–4 ms) ensures smooth syncing.

**Game Flow Highlights**

* Asteroids are spawned by the host only and synchronized using events.
* Bullet-asteroid collision triggers a scoring event, which is broadcast and acknowledged.
* Scores are tracked per NetworkID and updated consistently.
* Player disconnection triggers logging and can be enhanced for full rejoin support.

## 

## **(c) Individual contribution**

Tan Yong Chin (t.yongchin, 2301359):

Integrated game events and optimized the acknowledgment (ACK) system.

Thang Weng Khong (wengkhong.thang, 2301372):

Designed and implemented the Lock-Step ACK and ReACK system; integrated game events into the lock-step architecture.

Irwen Yap Zi Yang (y.ziyangirwen, 2301345):

Handled engine setup, networking setup, and overall game structure and polish.

Wilmer Lee Jun Rong (wilmerjunrong.lee, 2200691):

Worked on specific event behaviors and integrated them into the existing network architecture.

### **Conclusion**

Spaceships delivers a synchronized multiplayer experience over LAN using a custom UDP-based protocol. Key gameplay elements like asteroid spawning, bullet firing, collisions, and scoring are synchronized using a lock-step system with acknowledgment handling. The game supports reconnections and maintains consistent state across all clients. Overall, it provides smooth, reliable multiplayer gameplay.