Spaceships Multiplayer Game

- Design Report

Members:

Bryan Ang Wei Ze (2301397)

Tham Kang Ting (2301255)

Low Yue Jun (2301302)

Michael Henry Lazaroo (2301394)

Contents

1. Game Design Introduction

Spaceships is a multiplayer adaptation of our previous module's assignment. In this game, players control spaceships and navigate through a field of asteroids, shooting at them to score points while avoiding collisions. The multiplayer version requires four players to compete against each other over a local area network (LAN), bringing a competitive dimension to the traditional game.

Gameplay Mechanics

Core Mechanics

- **Ship Movement:** Players control ships with directional keys to accelerate forward/backward and rotate.
- **Shooting:** Player can fire bullets by pressing the spacebar key.
- **Asteroids:** Floating space rocks that move around with varying speeds and sizes.
- **Collision:** Ships can collide with incoming asteroids, resulting in loss of life and ship respawning.
- **Scoring:** Players earn 100 points for destroying each asteroid, first to 5000 points win.
- **Lives System:** Each player has limited lives, once all lives are used, it's deemed game over for that player. Alternatively, if only one player remains with lives, that player wins by default.

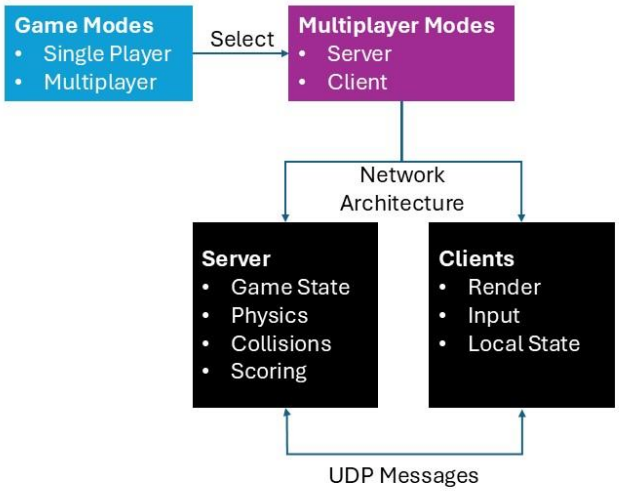
Multiplayer Mechanics

- **Player Identification**: Each player is assigned a unique ID (0-3) and is identified by highlighted yellow name
- **Real-time Synchronisation**: All game objects (ships, bullets, asteroids) are synchronized across all connected clients.
- **Score Tracking**: Each player's score is visible to all other players, creating a competitive environment.
- **Network Resilience**: The game can handle minor network disruptions and player disconnections without affecting other players.

2. Implementation Strategy

System Architecture

The Spaceships multiplayer implementation follows a client-server architecture, which provides a single source of truth for game state and simplifies synchronization.



Message Types

Client to Server	Server to Client
Join Request	Join Response
Player Input	Game Start
Object Destroy Ack	Game State Update
Ping	Object Create
Disconnect	Object Destroy
	Score Update
	Game End
	Pong
	Disconnect

Key Components

Game Mode Selection

- Abstract ***GameMode*** interface that defines common functionality.
- ***SinglePlayerMode*** for local gameplay.
- ***MultiPlayerMode*** for networked gameplay.
- Reads server IP from configuration file for client mode.

Network Layer

- UDP-based communication system
- Message serialization and deserialization
- Connection management

Server Implementation

- ***StandaloneServer*:** Manages game state, processes inputs and broadcasts updates
- Handles player connections, disconnections and timeouts
- Implements game logic, physics and collisions

Client Implementation

- ***MultiplayerMode*:** Processes network messages and updates local game state
- Sends player input to server
- Renders game state

Synchronization Mechanisms

- Object ID tracking: Each game object has a unique network ID
- Sequence numbering: Messages include sequence numbers for ordering
- Acknowledgments: Critical events require acknowledgment from clients
- Position/velocity interpolation: Smooth movement despite network jitter

Networking Details

Message Types:

The network protocol uses a message-based system with different message types for various game events:

Connection Messages:

- JOIN_REQUEST: Client requests to join server
- JOIN_RESPONSE: Server accepts/rejects join request

Game State Messages:

- GAME_START: Signals game start with seed
- GAME_STATE_UPDATE: Periodic updates of object positions
- SCORE_UPDATE: Updates player scores

Object Lifecycle Messages:

- OBJECT_CREATE: Creates new game objects
- OBJECT_DESTROY: Removes game objects
- OBJECT_DESTROY_ACK: Acknowledges destruction

Input Messages:

- PLAYER_INPUT: Transmits player control inputs

System Messages:

- PING/PONG: Network latency measurement
- DISCONNECT: Signals player/server disconnection
- GAME_END: Signals game over condition

Synchronization Strategy:

The game employs several synchronization mechanisms to ensure consistent gameplay across all clients:

Authoritative Server Model:

- Server is the authority for all game physics and object states
- Clients send inputs, server calculates resulting states
- Prevents cheating and ensures consistent game simulation

State Updates:

- Regular transmission of object positions, velocities, and orientations
- Updates prioritize important objects (player ships, bullets)
- State interpolation for smooth movement

Event Confirmation:

- Critical events (asteroid destruction) require acknowledgment
- Server waits for acknowledgments before finalizing events
- Ensures all clients see consistent game state for important events

Object Lifecycle Management:

- Objects are created/destroyed via explicit messages
- Network IDs ensure proper object tracking
- Object ownership attribution for scoring

Network Quality Monitoring:

- Regular ping/pong messages measure latency
- Clients display connection quality indicators
- Server tolerates temporary connection issues

3. Technical Challenges and Solutions

Challenge 1: Object Synchronization

Problem: Keeping asteroids and ships synchronized across all clients.

Solution: The server acts as the authority for all object positions and broadcasts regular state updates to all clients. Clients apply these updates to their local object instances, ensuring all players see the same game state.

Challenge 2: Event Timing

Problem: Ensuring events (like asteroid destruction) happen at the same time for all players.

Solution: The server manages all collision detection and broadcasts destruction events. Clients must acknowledge these events before they are finalized, creating a rudimentary lockstep for critical game events.

Challenge 3: Input Latency

Problem: Player inputs need to feel responsive despite network latency.

Solution: High-frequency communication combined with local physics interpolation provides smooth gameplay while maintaining server authority over game state.

Challenge 4: Network Reliability

Problem: UDP is unreliable, and packets can be lost.

Solution: Critical messages (like object destruction) use acknowledgment systems. State updates are frequent enough that occasional packet loss doesn't significantly impact gameplay.

Challenge 5: Game State Management

Problem: Maintaining a consistent game state across server and all clients.

Solution: Server maintains the authoritative state, with regular broadcasts to clients. Network IDs ensure proper object tracking, and sequence numbers prevent out-of-order message processing.