**Design Report – Networking Assignment 4**

**Multiplayer Asteroids Game**

Group Members

* Adam Goh Zheng Shan (2301303) – goh.a@digipen.edu
* Brandon Poon (2301224) – b.poon@digipen.edu
* Zulfami Ashrafi (2301298) – b.zulfamiashrafi@digipen.edu
* Gabriel Peh Jun Wei (2301454) – peh.j@digipen.edu
* Kuek Wei Jie (2301325) – k.weijie@digipen.edu

**1. Design Overview**

This project implements a real-time, multiplayer Asteroids game over a UDP networking system. Since UDP is inherently unreliable, we designed and implemented a lightweight synchronization and prediction system to ensure smooth, responsive gameplay while handling potential packet loss and data inconsistencies.

The objective of this design was to:

* Synchronize player movement, position, and actions (shooting bullets) across multiple clients.
* Compensate for packet loss, network delay, and out-of-order data.
* Maintain smooth gameplay and consistency across clients without relying on TCP.

**2. Data Format & Network Communication Protocol**

**2.1 Player Data Format**

Each player’s state is synchronized using a custom data packet format, containing:

* Player ID (int) – Unique identifier for each client.
* Position (Vector2) – Player’s current X, Y position.
* Velocity (Vector2) – Player’s movement velocity.
* Direction (float) – Player’s current rotation.
* Shoot Flag (bool) – Indicates if the player fired a bullet.
* Bullet Count (int) – Number of bullets currently active.

**2.2 Network Packet Flow**

The game uses UDP Broadcast for all communication:

1. Position Updates  
   Every client periodically broadcasts its current position, velocity, and direction to other players.
2. Bullet Events  
   When a player fires a bullet, a shoot event is sent to all clients, ensuring that all players see the same bullets.
3. Lag Compensation Data  
   Player movement updates include timestamped positions, allowing for basic interpolation and lag smoothing.

**3. Synchronization & Lag Handling**

To handle the unreliable nature of UDP and minimize the effects of packet loss, we implemented:

* State Interpolation  
  Each client uses Lerp-based smoothing between received positions and their own predicted state to reduce jitter and maintain smooth motion.
* Flag System  
  Each object instance is marked as either ACTIVE or LAGGING:
  + ACTIVE: Normal synchronization.
  + LAGGING: Detected high packet delay or packet loss triggers compensation logic.
* New Data Detection  
  A newDataReceived flag triggers interpolation updates when fresh data is received from the network.

4. Error Handling & Known Issues

A key challenge in our design was ensuring data integrity while avoiding vector out-of-bounds errors and invalid memory access.  
We implemented the following safeguards:

* Vector Index Validation:  
  Whenever accessing lerpData or currData, we ensure the index is valid and entries exist.
* Conditional Access:  
  All .find\_if() checks were updated with proper .end() checks before accessing the data.
* Bullet List Management:  
  Bullets are dynamically added and removed from bullet\_list with care taken to avoid accessing destroyed or non-existent bullets.
* Data Structure Consistency:  
  Player IDs and indices in the player\_list are verified before use to prevent out-of-range access.

**5. System Verification & Testing**

We conducted the following tests to verify system stability and reliability:

Functional Testing

- Verified that player movements and bullet firing actions were correctly synchronized across all connected clients.  
✅ Confirmed that asteroids and pickups behaved consistently on all clients.

Packet Loss Testing

- Simulated packet loss scenarios to test if lag compensation and interpolation mechanisms successfully smoothed movement without game-breaking issues.

Stress Testing

- Ran the game with four connected clients, spamming movement and shooting actions, to ensure no crashes or synchronization failures.

Debugging & Logs

- Used in-game debug prints and console logs to monitor player positions, bullet counts, and flag states.  
- Fixed critical vector subscript out of range bugs caused by invalid access to empty vectors.

**6. Individual Contributions**

Kuek Wei Jie (2301325, k.weijie@digipen.edu)

* Implemented lag compensation logic and LERP smoothing system.
* Managed player data structures and synchronization protocols.
* Optimized the interpolation system to prevent abrupt position jumps.

Adam Goh Zheng Shan (2301303, goh.a@digipen.edu)

* Set up the UDP networking layer and broadcast communication.
* Implemented shooting and bullet synchronization system.
* Handled dynamic object creation and destruction logic.

Zulfami Ashrafi (2301298, b.zulfamiashrafi@digipen.edu)

* Developed the player data format and packet handling routines.
* Managed bullet lifecycle and cleanup to avoid memory leaks.
* Contributed to lag detection and flag system.

Brandon Poon (2301224, b.poon@digipen.edu)

* Wrote comprehensive debug logs and assisted in troubleshooting synchronization errors.
* Conducted stress testing with multiple clients to ensure stability.
* Contributed to protocol design documentation.

Gabriel Peh Jun Wei (2301454, peh.j@digipen.edu)

* Simulated packet loss conditions to test the robustness of the compensation system.
* Implemented checks to handle packet duplication and out-of-order reception.
* Assisted in validating game state consistency across clients.

**7. Conclusion**

Through iterative development and testing, we successfully built a functional, real-time multiplayer Asteroids game over UDP. Our design demonstrates an understanding of the limitations of UDP and implements practical solutions for synchronization, lag handling, and packet loss resilience.