GSync v1 – Project Proposal

Real–Time Grid Clash over UDP

CSE361 Project (Phase 1)

Team Members

#	Name	Student ID
1	Member 1 Name	ID123456
2	Member 2 Name	ID123456
3	Member 3 Name	ID123456
4	Member 4 Name	ID123456
5	Member 5 Name	ID123456

1 Assigned Scenario

A lightweight real–time multiplayer system where multiple clients compete to "claim" cells on a shared 10×10 grid. The server maintains an authoritative view of the world and periodically broadcasts snapshots to all connected clients. Clients send unreliable "event" messages to request cell ownership.

2 Motivation

Traditional reliable transports (e.g., TCP) introduce head–of–line blocking, variable latency and retransmission delay, which are undesirable in real–time interactive systems. Instead, many modern games (Quake, Valorant, Roblox) use UDP with custom reliability strategies: periodic state replication, soft synchronization, timestamped events, and redundancy instead of ACKs.

This project explores these concepts in a controlled environment, focusing on:

- Low–latency state distribution
- Conflict resolution under race conditions
- Mitigating packet loss without TCP
- Deterministic server authority model

3 Proposed Protocol Approach

3.1 Transport

UDP sockets are used for all communication. No connection handshake is required; participants register via an `INIT` message.

3.2 Reliability Strategy

- No per–packet ACK for game state
- Server broadcasts snapshots at fixed rate (10 Hz default)
- Each packet contains up to the last 3 snapshots (redundancy window)
- Clients always apply the **newest valid snapshot** inside payload

3.3 Event Model

Clients issue `ACQUIRE_REQUEST` events to claim cells. Events are applied in arrival order (first–come–first–served). The server is authoritative and logs all accepted/rejected events.

3.4 Message Types

ID	Name	Direction	Purpose
0	SNAPSHOT	Server → Clients	Periodic state broadcast
1	EVENT	Client → Server	Cell acquisition request
2	ACK	(Phase 2)	Reliability diagnostics
3	INIT	Client → Server	Register player session

3.5 State Representation

The grid is encoded as a 100–byte array where each cell stores the owning player ID (0 = unclaimed). Snapshot payload:

[`grid_n (1B)` | `owners[100]`]

4 Feasibility & Current Progress

- Header format defined and implemented in both client and server
- CRC32 integrity check implemented
- Working UDP prototype with live pygame UI
- Snapshot redundancy ring buffer (size = 3)
- CSV logging of snapshots and events for analysis

4.1 Prototype Features

- Clients are auto–registered via `INIT`
- Server broadcasts real–time snapshots at 10 Hz
- Clients render a live grid UI and can click to claim cells
- Event resolution is deterministic and logged

5 Conclusion

The proposed architecture meets the scenario requirements while remaining extensible for later phases. The use of UDP with soft–state replication and redundancy allows the system to tolerate packet loss without complex retransmission logic, while still giving the server fully authoritative state control.