PROCESS SYNCHRONIZATION

IN MMORPG DUNGEON QUEUING

STDISCM S14

JP MARCELLANA

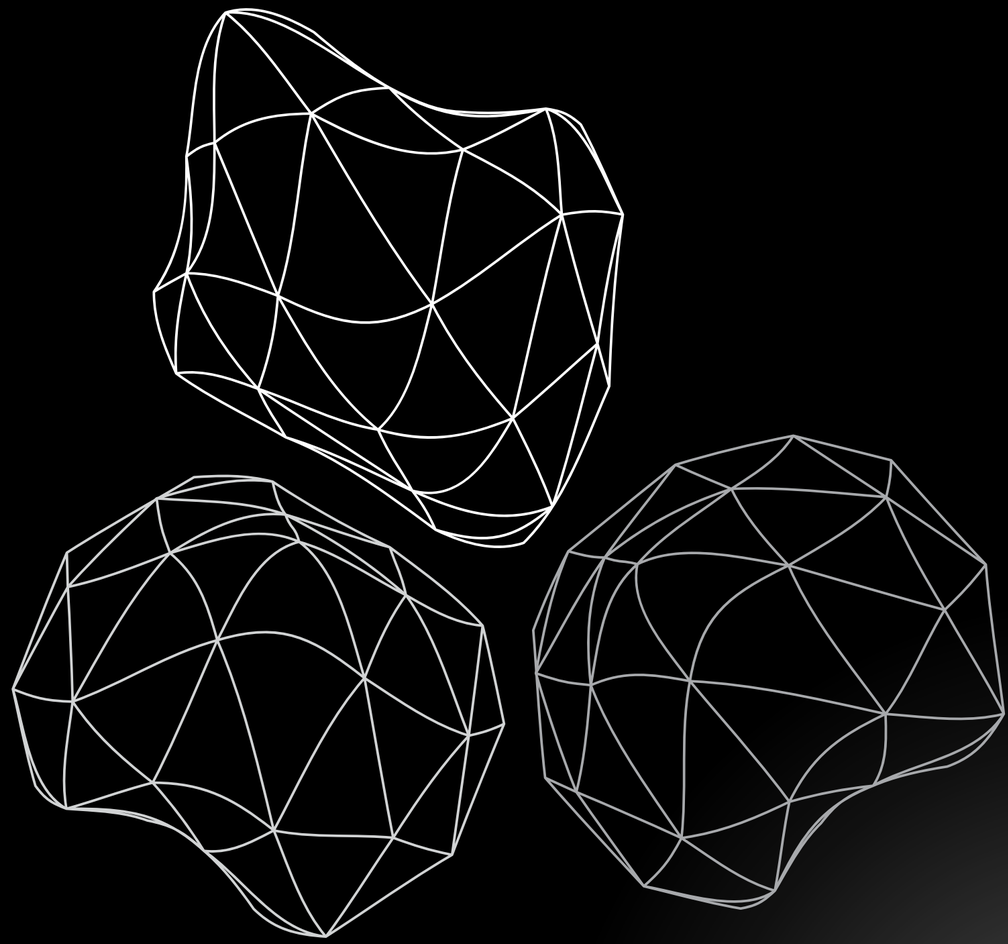
PROBLEM OVERVIEW

Objective: Manage dungeon queuing efficiently with process synchronizatio

Constraints:

- Max n concurrent dungeon instances
- Parties must contain 1 Tank, 1 Healer, 3 DPS
- Avoid deadlock & starvation
- Random dungeon duration t1 - t2 seconds

Solution: Implement thread synchronization for fairness & concurrency



DEADLOCK EXPLANATION

Deadlock occurs when two or more processes wait for each other indefinitely.

Potential Deadlock Scenario:

- A party starts forming but lacks a role (e.g., 1 healer missing)
- Other roles (Tank & DPS) remain locked, preventing new formations
- System halts as no party can proceed

Solution: Ensure resource availability before forming a party

STARVATION EXPLANATION

Starvation happens when a process waits indefinitely due to resource allocation priority.

Potential Starvation Scenario:

1. If priority is given to certain dungeon instances, some players may never get queued
2. If new players arrive while old ones are still waiting, some may be perpetually ignored

Solution: Fair queuing mechanism ensures all players get assigned fairly

SYNCHRONIZATION MECHANISMS USED

Mutex

- Prevents race conditions when modifying player counts & instances
- Ensures one thread modifies data at a time

Threading

- Enables concurrent dungeon instance handling
- Each instance runs on a separate thread

CODE IMPLEMENTATION OF SYNCHRONIZATION

Mutex Usage:

```
mutex mtx;
unique_lock<mutex> lock(mtx);
```

Randomized Dungeon Time Simulation:

```
// Simulate dungeon duration
int dungeonTime = getRandomNumber(minTime, maxTime);
this_thread::sleep_for(chrono::seconds(dungeonTime));
```

Threading for Dungeon Instances

```
vector<thread> threads;
    for(int i=0; i < maxConcurrentInstance; i++){
        threads.emplace_back(queueParties, i);
    }
```